Emmanuel Gaudin
Elie Najm
Rick Reed (Eds.)

LNCS 4745

SDL 2007: Design for Dependable Systems

13th International SDL Forum
Paris, France, September 2007
Proceedings

Springer

Emmanuel Gaudin Elie Najm Rick Reed (Eds.)

SDL 2007: Design for Dependable Systems

13th International SDL Forum
Paris, France, September 18-21, 2007
Proceedings

Springer

Volume Editors

Emmanuel Gaudin
PragmaDev SARL
18, rue des Tournelles, 75004 Paris, France
E-mail: emmanuel.gaudin@pragmadev.com

Elie Najm
ENST
Département Informatique et Réseaux
46, rue Barrault, 75634 Paris Cedex 13, France
E-mail: Elie.Najm@ENST.fr

Rick Reed
Telecommunications Software Engineering Limited
The Laurels, Victoria Road, Windermere, Cumbria LA23 2DL, United Kingdom
E-mail: rickreed@tseng.co.uk

Library of Congress Control Number: 2007934912

CR Subject Classification (1998): C.2, D.2, D.3, F.3, C.3, H.4

LNCS Sublibrary: SL 5 – Computer Communication Networks and Telecommunications

ISSN 0302-9743
ISBN-10 3-540-74983-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74983-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12161393 06/3180 5 4 3 2 1 0

Preface

This volume contains the papers presented at the 13^{th} SDL Forum, Paris, France entitled "Design for Dependable Systems" and reflects the intent to have a balance between experience reports and research papers related to System Design Languages.

The language that was at the heart of the first few SDL Forums was the ITU-T Specification and Description Language defined in Z.100, and the application domain was almost entirely fixed-line telephone communication. Mobile telephony was for the super-rich and electronics in cars was just for radios.

Ever since its inception, 30 years ago, the Z.100 language has been used for model-driven development in the telecommunication industry. Nowadays, model-driven engineering is a must for all industries and has been generalized by OMG to all application domains as covered by a paper on an automotive case study in this volume. What has been happening over the past few years is that the infrastructure has been put in place providing good support for the model-driven paradigm, so that the economic benefit of the approach makes it more of a necessity than a choice for designing dependable systems. The experience report from Motorola in this volume underlines this trend.

Although the SDL Forum Society that organizes these SDL Forums has it roots in telecommunications, the System Design Languages needed for modeling in that industry are applied in other real-time engineering domains such as aerospace, the ubiquitous Bluetooth devices, and railways. For the last few years all modeling languages and technologies have had a tendency to converge towards UML, and since UML 2.0 and its profile definition capability came out, there is now an amazing number of diverging profile proposals based on older technologies. This was reflected in the conference programme with tutorials on SysML, SDL-RT, MARTE, and Z.109 covering different aspects of system modeling. An example in this volume is the paper that utilizes the UML 2.0 Testing Profile.

This latter paper is one of a number that shows the continuing interest and developments in the ITU-T Testing and Test Control Notation (TTCN). Although much of the evolution of TTCN has been through the work of ETSI, it is still largely seen as an ITU-T standard. In some ways this makes sense as ITU-T re-publishes the ETSI revisions of TTCN as a truly international standard (Z.140 series). TTCN is widely used with the ITU-T Message Sequence Chart (Z.120) and Specification and Description Language (Z.100 series). These are also used with another ITU-T product, Abstract Syntax Notation One (X.680 series), which is used to define protocol data units with their associated encoding rules (X.690 series). However, these languages are not thought to be adequate to capture requirements. A new language for User Requirements Notation (Z.150 series) is in progress, which includes Use Case Maps — covered by another paper in this volume.

So with all these ITU-T languages for system design, what is the role of UML?

UML is seen, as its name implies, as a unifying concept between languages. Because UML leaves a number a semantic issues open and even states frequently that there is *no specific notation* for a particular concept, it is in reality largely a framework that has to be populated with specific semantics and notations before it can be used to completely develop products. One route is to choose a particular UML tool, whose implementation (such as writing actions in C or Java) will have fixed certain issues, but at the cost of potentially being locked into that tool. Another route is to provide UML profiles for existing languages, thus not only binding UML to the semantics and notation of the language, but also providing some glue between different notations. It is the latter route that the ITU-T is taking (albeit rather slowly), with Z.109 being approved in 2007 as the UML profile for Z.100. Other profiles are in the ITU-T work plan for X.680, Z.120, Z.140 and Z.150. A related path is presented in the first paper in the volume, providing a meta-model for (a subset of) Z.100.

UML also has another role. If you ask someone who claims to be using UML which diagrams they use, often the reply will be that they mainly use Class Diagrams and Object Diagrams. The other 11 types of UML diagrams are used less frequently and some quite rarely (if at all). This is partly because the Class Diagrams and Object Diagrams meet a need that is not well met by other notations. Even the ITU-T in its 1996 Z.100 SDL+ methodology supplement suggested using diagrams in the Object-Modeling Technique notation (a forerunner of UML subsumed into UML in the unifying process). This is why it is natural to use these diagrams with the ITU-T languages: UML is frequently used for class and object modeling with Z.100 and other state machine languages in this volume and elsewhere. UML therefore not only provides the glue, but itself provides an important member of a set of System Design Languages.

Although the original Z.100 of 30 years ago was a paper and pencil language, none of this engineering today would be practical without computer-based tools because the systems in question are much more complex. This is evident from most of papers. As well as tools to directly support System Design Languages, included in this volume are papers on a real-time operating system and the use of probability modeling to analyze realistic-size networks without encountering state space explosion. At first glance, it may seem that these papers are not relevant, but you will probably change your mind when you read the papers, as a key issue in both cases is performance. There are many factors involved in the design for dependable real-time systems, so it is hard to predict what might be relevant for a future SDL Forum.

Thanks

A volume such as this could not, of course, exist without the contributions of the authors, who are thanked for their work.

The Programme Committee were also the reviewers of the papers, and are thanked for their work selecting the papers and the programme.

Irfan Hamid of ENST is thanked for his editorial assistance in preparing this volume.

The organization was greatly assisted by the various sponsors that provided valuable support. SDL 2007 was sponsored by:

- Centre National de la Recherche Scientifique
- Cinderella
- France Telecom
- PragmaDev
- Télécom Paris - École Nationale Supérieure des Télécommunications (ENST)
- Telelogic

July 2007

Emmanuel Gaudin
Elie Najm
Rick Reed

Organization

Each SDL Forum is organized by the SDL Forum Society with the help of local organizers. The Organizing Committee consists of the Board of the SDL Forum Society plus the local organizers and others as needed depending on the actual event. For SDL 2007 the local organizers from PragmaDev and ENST need to be thanked for their effort to ensure that everything was in place for the presentation of the papers in this volume.

Organizing Committee

Chairman, SDL Forum Society	Rick Reed (TSE Ltd.)
Treasurer, SDL Forum Society	Martin von Löwis (Hasso-Plattner-Institut)
Secretary, SDL Forum Society	Andreas Prinz (Agder University College)
Conference Chair	Emmanuel Gaudin (PragmaDev)
Programme Committee Chair	Elie Najm (ENST)

Programme Committee

Daniel Amyot (Université d'Ottawa, Canada)
Reibert Arbring (Ericsson, Sweden)
Rolv Bræk (NTNU, Norway)
Eric Brunel (PragmaDev, France)
Pierre Combes (France Telecom, France)
Philippe Desfray (Objecteering Software, France)
Laurent Doldi (Isoscope, France)
Anders Ek (Telelogic, Sweden)
Jaqueline Floch (SINTEF, Norway)
Birgit Geppert (Avaya Labs Research, USA)
Reinhard Gotzhein (Universität Kaiserslautern, Germany)
Jens Grabowski (University of Göttingen, Germany)
Susanne Graf (Verimag, France)
Peter Graubmann (Siemens, Germany)
Loïc Hélouët (INRIA Rennes, France)
Paul Herber (Sandrila, UK)
Dieter Hogrefe (ETSI - MTS, Germany)
Eckhardt Holz (University of Potsdam, Germany)
Ferhat Khendek (Concordia University, Canada)
Tae-Hyong, Kim, KIT, Korea)
Shashi Kumar (Jönköping University, Sweden)
Philippe Leblanc (Telelogic, France)

SDL Forum Society

The SDL Forum Society is a not-for-profit organization that in addition to running the SDL Forum:

- Runs the SAM (System Analysis and Modeling) workshop every 2 years between SDL Forum years.
- Is a body recognized by ITU-T as co-developing the Z.100 to Z.109 and Z.120 to Z.129 and other language standards;
- Promotes the ITU-T System Design Languages.

For more information on the SDL Forum Society, see `www.sdl-forum.org`.

Table of Contents

A Model-Based Standard for SDL

Andreas Prinz[1], Markus Scheidgen[2], and Merete S. Tveit[1]

[1] Faculty of Engineering, Agder University College
Grooseveien 36, N-4876 Grimstad, Norway
{andreas.prinz,merete.s.tveit}@hia.no

[2] Department of Computer Science, Humboldt Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
scheidge@informatik.hu-berlin.de

Abstract. Language descriptions have much information captured in plain (English) text, and even the formalised parts are often informally connected with the overall language definition. These imprecise descriptions are hardly usable to automatically generate language tool environments out of the language standard. SDL has already managed to define syntax and semantics in a quite formal way. Currently, this formality is connected by using different types of grammars. Meta-models, however, have proven to be a good way of expressing complex facts and relations. Moreover, there are tools and technologies available realising all language aspects based on completely formal and still easily understandable meta-model-based descriptions. This paper is about an experiment of combining all these existing techniques to create a definition of (a subset of) SDL. This allows to have immediate tool support for the language. This experiment includes the language aspects concrete syntax representation, static semantic constraints, and language behaviour. It turns out that this is almost possible.

1 Introduction

Model Driven Development (MDD) uses models to describe systems on a higher level of abstraction. This abstraction, i.e. hiding of much detail, is possible because models are instances of more and more complex modelling languages, which provide more and more specific concepts. Therefore, there is a need for more complex and (domain) specific modelling languages. Furthermore languages in an MDD environment are only meaningful if they come with a comprehensive tool environment. So there are two challenges: creating a human readable language standard and providing tool support for the language.

It is obvious that it is necessary to have a description of the language first. We will call such a description a meta-model. Today, there are several language description techniques and meta-tools that allow to describe and realise single language aspects like concrete syntax, static semantic analysis, model execution, or code generation. Tooling can be achieved by manually building language tools or by creating modelling tools automatically from the language description. In the latter case, the language description has to be completely formal.

The contribution of this paper is a combination of existing and new techniques forming one cohesive language description with the possibility to create a complete tool environment from it. We start with a representative sub-set of SDL (this sub-set provides all features necessary for the well-known camera example), and create meta-model-based descriptions that can function as a human-readable standard and an SDL tool environment including textual and graphical editors, static semantic checker, and model simulator.

The current SDL standard [8] with its formal semantics specification [7] already showed that most language aspects can be described formally without ambiguities. In [3], Prinz et. al. showed that even aspects that are usually described informally, like language behaviour, can be described formally allowing tools to be created from such formal descriptions in an at least semi-automated way. In [4] we discussed the possibility to use meta-modelling as the basis for integrating different languages and tools with each other. We already successfully evaluated the possibilities for automated tool support based on meta-models in the context of domain specific languages in [10,14].

In this paper we explain how different meta-modelling techniques work together. We focus on the two main purposes given above: how to present the language description in a user-friendly way and how to use the description for generating tools. Although in an ideal world, these two purposes would coincide, we could not achieve a complete match in this experiment.

The paper is structured as follows. In Sect. 2 we will introduce the different language aspects that we used in this experiment together with their relation to each other. The subsequent sections will present the approaches and technologies that we used to describe the different aspects one by one, namely structure (Sect. 3), constraints (Sect. 4), representation (Sect. 5), and behaviour (Sect. 6). Each section contains parts of the SDL language as examples. In the concluding Sect. 7, we discuss our results and suggest further work.

2 Basics

In [9] meta-modelling is defined as: *The construction of an object-oriented model of the abstract syntax of a language.* However, in our article we use the term meta-model in a wider sense: *A meta-model is a model that defines a language completely including the concrete syntax, abstract syntax and semantics.*

As a language description, meta-models can have several aspects that we have already identified in [10]. Figure 1 shows these aspects. Even though there is no complete agreement about what parts a language description consists of, these or similar parts can be identified in most contexts. The picture shows the following parts.

Structural information for the meta-model includes all the information about which concepts exist in the domain and how they are related. An example of this would be a MOF (Meta Object Facility) class diagram. In our understanding, this part does just include very simple structural properties and not more advanced concepts that rely on the use of constraints.

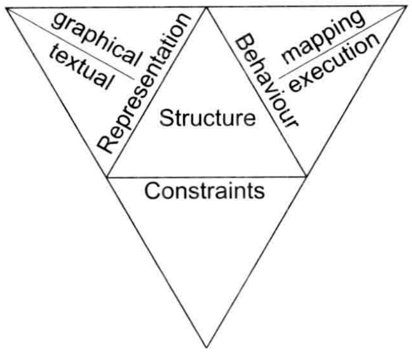

Fig. 1. Structure of a Meta-model

Constraints give additional information about the structure in that they identify the allowed structure according to additional logical constraints. This will include first-order logic constraints (e.g. written in Object Constraint Language (OCL)) as well as multiplicity constraints. In classical compiler theory these are collected under the name of static semantics and in a meta-model context they are called well-formedness rules.

Representation describes model serialization syntax and information about how the models are to be (re)presented to the user. The textual grammars (concrete textual syntax) are well understood in terms of compiler theory. When it comes to graphical grammar (concrete visual syntax), there is less agreement and there are some open research topics.

Behaviour describes how the model is used. This item includes execution of the model as well as mappings. By mapping we understand a relation between the model itself and another representation, e.g. in another language. A typical example would be a compiler from Java to JVM, or a mapping from a platform independent model to a platform specific model. An execution is the real run of the model, which is of course only possible if the model is executable. A typical example here would be a run of a Petri net.

In Fig. 1, the structure is the central aspect and all the other parts relate to the structure. The constraints have to be connected to the structural elements that they constrain. The representation parts describe the representation of elements in the structure, whereas the behaviour parts describe a behaviour for the elements defined in structure.

3 Structure

The structure part of a language description defines an abstract data structure for models, programs, or specifications written in that language. Like in model-driven development, object-oriented models in the form of class diagrams, are used in most meta-modelling architectures to model structures. These type models use classes as refinable classifications of entities by means of shared characteristics, modelled with attributes. Associations are used to classify the relations between entities. Associations are just a special kind of classifiers and actual links are a special kind of entities.

For a user-friendly description of the language, we use CMOF from MOF 2.0 [11]. CMOF (complete MOF) provides additional concepts to model abstractions compared to EMOF (essential MOF also part of MOF 2.0), which only defines a set of basic meta-modelling features. Examples for these additional CMOF features are property refinements, which allow to relate attributes or association ends in the context of classifier specialisation.

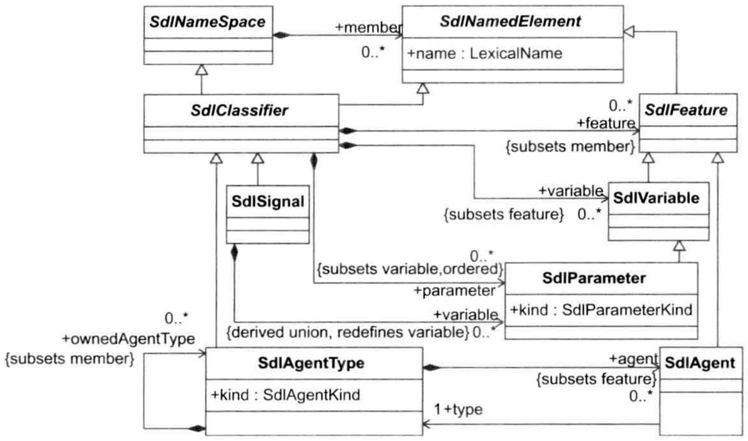

Fig. 2. Classifier concepts in SDL

In terms of languages, classes classify model elements based on the language concept that they instantiate. For example, all the agent types in all the existing SDL specifications are instances of the agent type concept. The first sample meta-model part in Fig. 2 describes the language concept *agent type* as a meta-model class. Attributes and associations are used to define the structural characteristics of agent types: an agent type can contain other agent types, it can contain type-based agents, it has parameters and variables. The example also shows how characteristics of more abstract language concepts can be reused. Agent types and signals for example, are just special SDL classifiers. SDL classifiers have features, like variables or parameters, as general characteristics. Variables are just one special form of features, and parameters just one special form of variables. Agent types inherit containment of variables and parameters and extend their set of features, containing variables and parameters already, with agents as just another type of feature. Signals inherit ownership of parameters. Signals also inherit ownership of variables and features, but only allow parameters as possible variables or features. The redefinition of the property variable in signal as a derived union ensures that parameters are the only possible subset of variables.

Using all the CMOF features to express abstractions, enabled us to compose a meta-model for SDL from a predefined library of abstract language concepts. We re-used the UML infrastructure library accordingly to create the SDL meta-model. The UML infrastructure library was used to define the UML, so this approach makes sure that the two languages UML and SDL have a common base in their underlying language infrastructure.

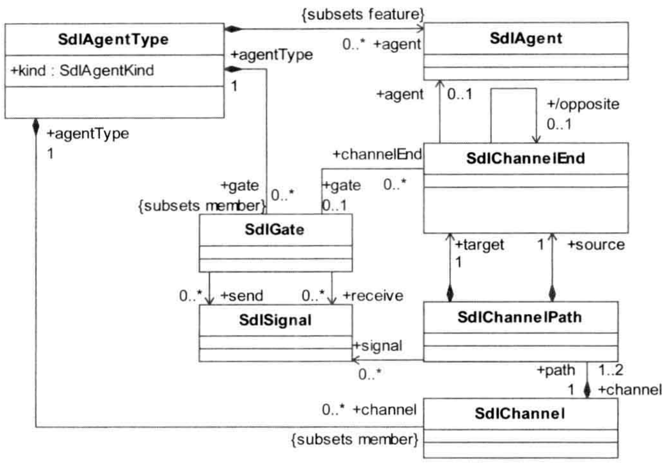

Fig. 3. Concepts for communication structures in SDL

Figure 3 shows another part of the SDL structure meta-model. This meta-model part covers the concepts of the sample specification in Fig. 4, which shows a block type definition (the entire example can be found in [15]). This block type definition is shown twice: in SDL syntax and as an object diagram, instantiating the SDL structure meta-model.

After having defined the structural meta-model in MOF 2.0, we had to find a proper tool supporting such descriptions. Although it is possible to find tools for MOF 2.0 (e.g. [13]), we decided to take a simpler tool which allows better integration with the other aspects as described in the next sections.

For the language tooling, we use Ecore (the meta-modelling language of EMF [1]). Ecore is a simple language allowing to express structures with just a few basic concepts. It is similar to EMOF. In Ecore the expressive power of the CMOF additional concepts has to be implemented manually, for example with OCL-expressions. Because of its simplicity, Ecore has the advantage of a clearer mapping to programming languages and more extensive tool support.

Compared with the SDL standard, the MOF-based structure definition yields almost the same object structure of a specification. The advantage is that it has much richer classification of the language concepts.

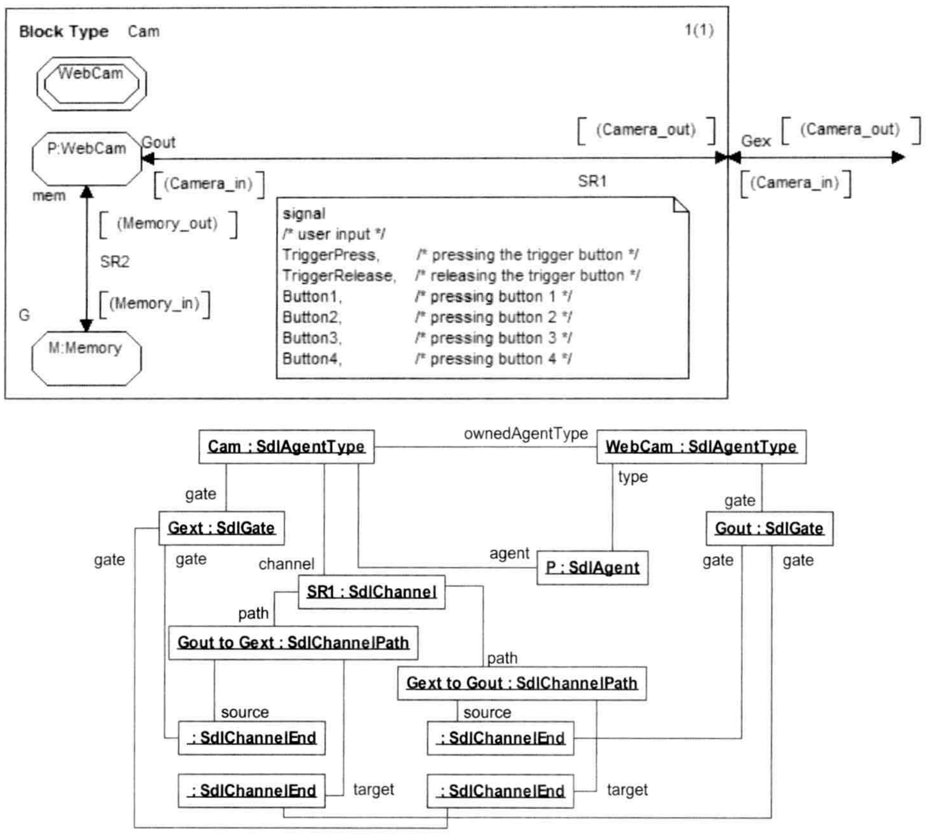

Fig. 4. Part of an example SDL specification and its model representation

4 Constraints

Structure models are designed to define valid graphs of objects and links, by defining classes and associations. To complement these concepts of constructive modelling, we use boolean expressions to constrain the possible instances of a meta-model.

To define such static semantic rules for SDL, we use the Object Constraint Language (OCL) [12]. OCL is specifically designed as an expression language for object-oriented structures. It allows to define expressions based on types defined in a meta-model. These expressions, defined at meta-level, can then be evaluated on models. OCL is a statically typed language. Each formula is defined in the context of a meta-model type. Based on this context type an expression can use the features of the corresponding meta-element to navigate through models. OCL uses several predefined operators and functions to combine feature values into