14

Improving the Development and Validation of Viewpoint Specifications

N. Fischbeck, J. Fischer, E. Holz, O. Kath, M. Löwis, R. Schröder
Humboldt-Universität zu Berlin, Dept. of Computer Science
Axel-Springer-Str. 54a, 10099 Berlin, Germany
{holz/kath}@informatik.hu-berlin.de

Abstract

This paper presents an overview of the application of ITU-SDL for the development of distributed systems according to the RM-ODP. The use of the advanced SITE tools for the simulation and prototype code generation of/from SDL specifications of computational as well as engineering objects is illustrated. The advantages of this approach are shown by applying it on the ODP trading function as an example. Furthermore, concepts and principles for a smooth, semi-automatic SDL based transition between viewpoints, which are currently under development, are sketched out.

Keywords

RM-ODP, viewpoint, FDT, viewpoint languages, formal behaviour description, computational languages, TINA-ODL, CORBA-IDL, trading function, SDL-92

1 INTRODUCTION

Information technology systems are increasingly restructured as networked solutions to reap the benefit of cheaper and more flexible technology. Driven by the changes of the markets and the technology such systems are becoming more and more heterogeneous.

The development of the Reference Model for Open Distributed Processing (RM-ODP) is an ongoing joint standardization activity of ITU-T and ISO (ITU-T, 1995a). It aims at a framework to organize services within autonomous systems in order to facilitate interworking of software components distributed into larger and larger systems. The RM-ODP provides:

- general modelling approach and concepts and
- a method to divide a specification into viewpoints in order to simplify the specification process.

Although still in development, the ODP standard has already influenced major developments in the area of distributed and telecommunication systems. Two leading industrial consortia (OMG and TINA-C (TINA-C,1994)) have set up liaisons with the ODP standardization.

The viewpoint concept is essential for the whole RM-ODP. Each viewpoint is focusing on a subset of the properties of the system. The viewpoint concept can be applied at large at the whole system or at a different level of abstraction to individual components of the system. The RM-ODP does not define a general mapping or correspondence between every pair of viewpoint languages, however, special relationships between the computational and information resp. the computational and engineering viewpoint are identified. The RM-ODP does also not prescribe any chronological order for the development of specifications for the different viewpoints, on the contrary - it favours a parallel development of the specifications. Nevertheless, a development process starting with the enterprise specification, followed by (possibly interleaved) specification of information, computational and engineering viewpoints and concluded by the technology specification seems to be an adequate way for the development of ODP systems.

Part 3 of the RM-ODP (ITU-T, 1995c) contains the abstract definition of the viewpoint languages. A concrete syntax for these languages is not given, the language definitions contain only the concepts and rules for the specification from the selected viewpoint. This allows to use existing or evolving languages/notation techniques as concrete viewpoint languages. The languages envisaged here range from natural languages over programming languages to formal description techniques (FDT). Initial mappings between the computational and information languages and the standardized FDTs SDL, LOTOS, Estelle, Z and ACT.ONE are given in Part 4 of the RM-ODP (ITU-T, 1995d).

2 THE COMPUTATIONAL LANGUAGE

Seen from the computational viewpoint an ODP system can be viewed as a configuration of interacting (computational) objects which are supported by an infrastructure. The computational language is used to clearly identify in a

distribution transparent manner the objects within the system, the internal activities of these objects and the interactions between these objects. The computational language imposes structuring rules on the objects and interfaces. These rules specify when and how two or more objects may interact, they give the set of actions a computational object can engage in, they define the set of failures and errors which can occur and finally they provide a base for portability. A computational specification is a specification whose objects all have environment constraints and own only computational interfaces.

The computational language distinguishes between two main kinds of (computational) interfaces, Operational interfaces and Stream interfaces. An operational interface is described by a set of operations where an operation is a single headed action which is part of the behaviour of the server object. The head of the action is the operation invocation according to the invocation template of the operation signature. The tail(s) of the action are terminations according to the termination templates of the operation signature.

In order for two computational objects to interact, first their interfaces have to be bound by an implicit or explicit binding. Implicit binding is available only for operational interfaces whereas explicit binding can be used for operational as well as for stream interfaces. A binding must satisfy the interface binding rules, which prescribe constraints on the types of interfaces which are permitted to be bound. One of these constraints for operational interfaces is the requirement that one interface must be a client interface and the other one a server interface. A successful explicit binding action results in a binding object, which controls then the execution of operations at the bound interfaces. This comprises also operations to change the set of interfaces involved in the binding.

The behaviour of a computational object is specified by the activities that occur within the object (internal) and the interactions at its interfaces (observable behaviour). Templates are specifications of the common features of a collection of objects, interfaces or actions and are the base for an instantiation.

3 Concrete Viewpoint Languages

With the application of FDTs as concrete viewpoint languages the full range of methodologies and tools for the development and analysis of specifications as well as for the implementation support is open to the designer of ODP systems and components. This comprises besides textual/graphical editors tools for the formal verification and for the simulation of specifications.Moreover, designers of distributed or telecommunication systems are often familiar with at least one of the FDTs which makes a transition to ODP more seamless.

Unfortunately none of the FDTs does support directly the ODP concepts by its semantic core. Therefore a modelling of the ODP concepts by composing basic concepts of the FDT is required, what may lead to a quite substantial overhead. In general it is also not possible to define a library of ODP concepts, which could be used to develop viewpoint specifications using the different FDTs. Instead Part 4 gives a set of patterns how to write down specification elements and how to combine those elements in order to obtain a complete viewpoint specification. A (formal)

verification whether or not a specification is an ODP conform viewpoint specification is difficult or even impossible. Another shortcoming of the FDTs is their lack of object-orientation. Even SDL, which has adopted object-oriented concepts with the last language revision does not fulfil the requirements of ODP. This introduces again additional overhead to model ODP's object-oriented features.

Although in some cases the amount of specification to model basic ODP concepts may be large and therefore clutter the whole specification, the FDTs seem to be one appropriate means for the formal notation of viewpoint specifications. The best suiting language for the information viewpoint is Z, whereas SDL and LOTOS are expected to be key candidates for the computational viewpoint.

Interface and object description languages bridge the gap between (computational) specification and implementation. Different kinds of such languages have been developed by industry consortia, among them CORBA/ODP-IDL (ISO/IEC, 1996) and TINA-ODL (TINA-C, 1996). These languages all share the ability to describe the structure of objects and the signature of operations/services provided by objects in (programming) language independent terms.

IDL and ODL are well integrated in the application development process. Bindings to different programming languages (e.g. C++, Smalltalk, Eiffel) enable the automatic generation of stubs (client side) and implementation skeletons (server side). These skeletons have to be enriched manually (according to the viewpoint specifications) to obtain the complete object implementation.

ODL has been developed by TINA-C and is based on OMG-IDL. It extends IDL by providing support for those TINA concepts which are not part of the CORBA framework, e.g. streams, QoS-attributes, objects and object groups, objects with multiple interfaces. Moreover, an ODL specification describes not only the signatures of objects, interfaces and operations, it also specifies the required behaviour of an object or an interface and gives usage rules. However, these behaviour specifications can currently be given informally only. A detailed formal syntax for the behaviour specification is subject for further studies within TINA-C.

Although IDL and ODL both directly reflect most of the structural concepts of the computational language, they are not sufficient enough to check interoperability and conformance to standards. This is due to their missing abilities for an unambiguous formal behaviour specification. All conformance investigations are limited to a syntactical matching of object and interface signatures.

OMT is a further technique which has been applied for the development of viewpoint specifications, especially in the information viewpoint. However, similar to the interface/object description languages, only structural concepts are expressed with an OMT object model, the use of the dynamic and functional model (behavioural aspects) within the scope of ODP specifications occurs rather seldom.

4 Combined Approach

As it has been explained in the previous section, none of the languages and notation techniques currently applied is sufficient enough to be applied throughout different viewpoints. It is in general even not possible to express all the concepts of a single viewpoint (as they are defined in the RM-ODP) with one of the concrete techniques.

On the other side the application of existing techniques bears the advantage of the availability of design and validation/verification methodologies and supporting tools and the familiarity of the designers with these tools.

An approach to overcome the shortcomings of the different single techniques is therefore their combination. This increases not only the expressiveness of the specifications within on viewpoint but eases also the transition between different viewpoints (e.g. information to computational viewpoint, computational viewpoint to engineering viewpoint).

As FDT SDL has been selected as one of the most widely applied FDTs. The object-oriented features of SDL allow for a simple and appropriate modelling of entities. Within a joint project between Humboldt-University and the European Telecommunications Standardization Institute (ETSI) for the specification and simulation of large scale protocols like Broadband ISDN Signalling Protocol Q.2971 the advantages of object-oriented features in SDL have been proven (ETSI, 1997).

4.1 Combination of ODL/IDL and SDL

The main idea of this combination is the use of IDL or ODL as the initial computational language. The overall structure of the application is specified in terms of objects and their interfaces. An interface specification consist of the signatures of the operations provided by the interface, the exceptions which may be thrown and the attributes of the interface. Behavioural aspects are not included here (or may be given by informal text only). The next step is the transformation of these specifications into SDL. The result is an SDL-skeleton specification containing definitions of block types, (empty) process types and remote procedure specifications(signatures only).

In order to obtain a complete computational specification, the process types have to be complemented by the behaviour description (process graph specifying actions/ interactions). Such a computational specification serves then as input for the various tools available for SDL (simulation, symbolic execution, prototype generation).

The transformation of IDL and ODL to SDL is a rather straightforward mapping, the general principle is depicted in Figure 1, a detailed description is given in (Born, 1996). As it can be seen, the structure of the resulting SDL specification follows the rules defined in Part 4. A subset of the transformation rules concerning ODP-IDL was incorporated into the Part 4 standard. To automatize the transformation an prototype of an ODL2SDL translator has been implemented (Kolberg, 1997)

The main goal of the simulation of the final computational SDL specification is the validation of the functionality. The Integrated SDL environment SITE (SITE, 1997), which was developed at the Humboldt-University does support the analysis of an SDL specification by providing:

- Syntactical analysis
- Semantics analysis
- Generation of code (C++, Java)
- Simulation run-time library
- Prototype run-time library.

An overview of the SITE tools is given in Figure 2. The toolset does support the full range of the object-oriented features of SDL. The main difference of SITE

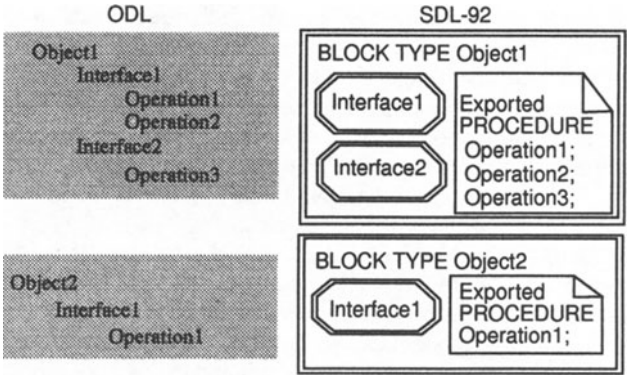

Figure 1 ODL to SDL mapping

compared to other SDL tools, is that the generated C++ code is not only structural equivalent to the SDL specification but can be used for simulation purposes as well as an implementation prototype. The purpose of the simulation is here the analysis

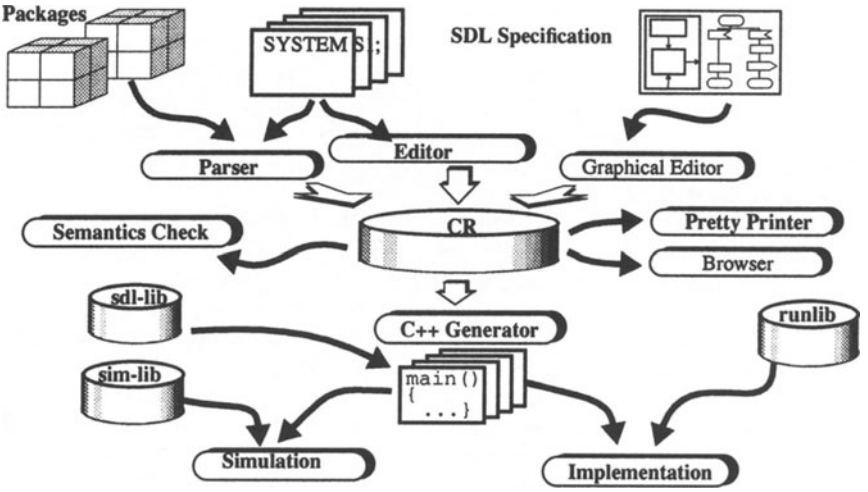

Figure 2 SITE structure

of the overall functionality of the specification at an computational level and independently from distribution aspects. The simulation of the computational specification is one aspect in the verification process. Because of the amount of data available from simulations, several techniques can be exploited to analyse these data, including analysis of a specific scenario, statistical analysis over a series of experiments, and the generation of message sequence charts. In order to support different analysis methods, the simulator is itself modular and provides interfaces to external analysis tools (Fischbeck, 1996). Since correctness and performance data depend partially on characteristics of engineering aspects (such as network delays),

the simulation model can be extended with such information. As the simulated system is separated from the physical system, the simulation itself is not executed in a distributed fashion.

In contrast to the simulation the prototype implementation takes distribution aspects into account. The unit of distribution is an SDL system, i.e. each SDL system is translated into an executable object. The communication between the single systems is based on OMG CORBA. In the sense of TINA an SDL system corresponds to an engineering computational object (eCO).

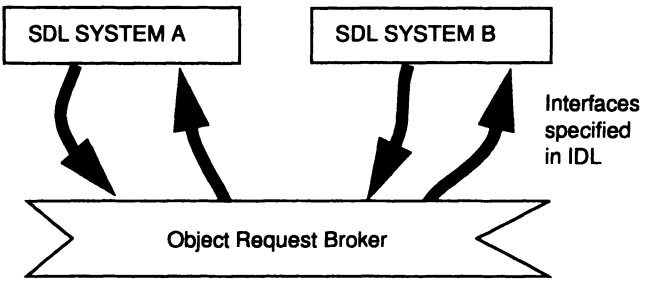

Figure 3 Corba based Prototype Implementation

Using the prototype the effects of the object distribution and of the underlying communication infrastructure on the functionality of the application can be studied at an computational level, which will also support the later development of an engineering specification. This approach has also been successfully used for the implementation of real applications (Fischbeck, 1996), using a proprietary communication platform instead of Corba.

4.2 Combination of OMT and SDL

The combination of SDL and OMT has been investigated within different projects (Holz, 1996b), (Guo, 1995) ranging from a translation of an OMT model to an SDL specification (including behaviour descriptions) to a pure linking mechanism between OMT class definitions to SDL data types. Especially the latter case is also supported by the commercial SDL tools, however caused by its restriction to data types it is limited to the information viewpoint.

Within the INSYDE (Holz, 1996b) project a three-stage methodology has been proposed. The first stage is the requirements capture and analysis. The Object Modelling Technique (OMT) as defined by Rumbaugh will be applied here. The second stage is the design phase, it is again split into two sub-phases: System Design and Detailed Design. For the System Design a special sub-set of OMT called OMT* has been defined. In contrast to OMT this sub-set has a well defined syntax and a transformational semantics. It is tailored towards the two target domains hardware and software. The Detailed Design uses the domain specific languages SDL for the specification of software. The initial specification for the detailed design is derived from the system design model by a semi-automatic translation of OMT* into SDL. In the final phase of the methodology a validation of the specifications of the hybrid system will be performed by simulation The methodology is supported by a

prototype toolset consisting partly of off-the-shelf tools and partly by new developed components to bridge the gap between the different stages.

5 APPLICATION OF THE TECHNOLOGY

In order to examine the proposed combination technology for the development of computational specifications the specification of the Trading Function (ITU-T, 1995e) has been chosen as an example by taking the IDL specification of the Trader as starting point. The main specification steps performed for this task were (Niedrig, 1996):

- Mapping
  The trader computational template described in OMG-IDL including its interfaces, attributes and operations (with their signatures and exceptions) is mapped to SDL by the IDL2SDL tool. As a result an SDL specification was obtained, which reflects the original structure as given in the ODP standard: A definition of an SDL block type (trading object template) including process type definitions (interface type definitions) with remote procedure definitions (operations) and their signatures and exceptions.
- Addition of behaviour specification
  The generated SDL specification does not contain any behaviour description, i.e. all process type definitions are empty and all procedure definitions are virtual and empty definitions. The addition of the behaviour description follows the informal specification given in the standard and is expressed in terms of actions and interactions at interfaces. During this step also additional interfaces to other ODP services have been identified as necessary for the provision of the trading function (i.e. Type Management, Security Management). The result of this step is a computational specification of the Trader. A direct validation/verification of this specification in isolation is, however, not possible. This is due toe the missing specification of other ODP services, which appear here still in a generic form.
- Adding an environment model & simulation
  The environment model completes the specification by importer and exporter objects and provides also access to the additional ODP services identified in the previous step. Within our work these services have been emulated by a very simple implementation or by a no-implementation (i.e. operations return a predefined value).

The final specification was used to derive conformance test cases for the trader. Part of this work was submitted to the trader standardization.

6 SUPPORTING DIFFERENT VIEWPOINTS

The positive results of the combination of different techniques within one viewpoint have encouraged further investigations on how to exploit such combinations also for a smooth transition between different viewpoints. The final goal of this development

is to provide tool support for the stepwise transition from one viewpoint to another by keeping at the same time consistency between specification at different viewpoints. A schematic view of such a development methodology is given in Figure 4.

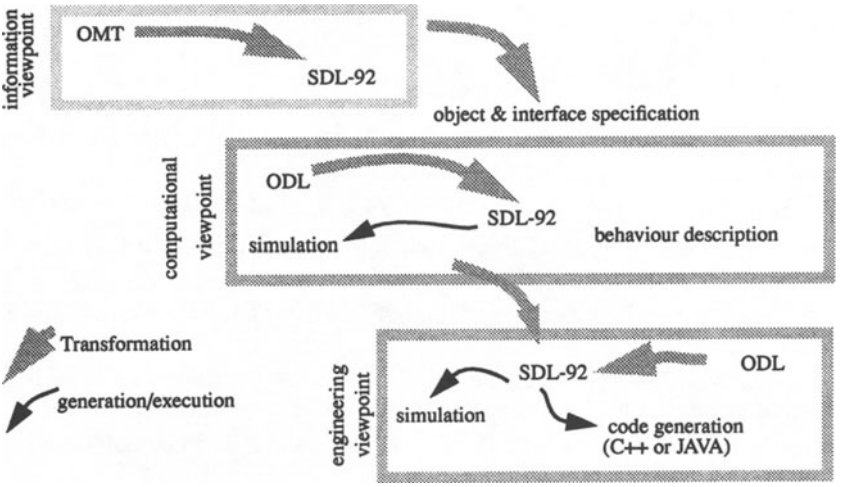

Figure 4 Transition steps between viewpoints

Because of the expected massive problems of transitions between computational and engineering specifications of objects and interfaces, the main focus of this development is the transformation of SDL computational descriptions into engineering ones. This approach will allow the simulation of engineering specifications under various distribution conditions as well as the generation of run-time code, done by the SITE tools. One main aspect of this transition is the semi-automatic generation and addition of engineering concepts to the computational SDL specification, e.g. stub, binder and object life cycle interface specifications for each object, done with ODL/SDL, too.

Another very important step in this development is to transit information specifications to analogous computational specifications, e.g. ODL descriptions with SDL skeletons.

Together these transformation concepts will bring an added value for the safe and fast development of software for large, distributed systems.

7 CONCLUSION

A combination of different specification and notation techniques has been shown as a feasible approach for the improvement of the development process. The main advantage lies in the availability of tools for existing techniques and the familiarity of the users with the techniques and tools. By combining the different techniques the

strength of the single techniques can be exploited and there limitations can be evaded. A tool support is a necessary precondition.

The exploitation of the results for the support of the transition between different viewpoints will be investigated further and in more detail in the ongoing project CAMOUFLAGE (Holz, 1996; Kath 1996).

8 REFERENCES

Born, M.; Winkler, M. (1996) SDL-92, CORBA-IDL und TINA-ODL im ODP-Entwurfsprozeß, Master Thesis, Humboldt-Univ. Berlin

ETSI Project Team 87(1997) Formal specification (SDL) for B-ISDN DSS2 CS2

Fischbeck, N.; von Löwis, M. (1996) Abschlußbericht TERMO: Entwicklung einer Nutzerschnittstelle zur Simulation von SDL-Systemen

Guo, F.; MacKenzie, T. (1995) Translation of OMT to SDL-92, Proc. of SDL'95, Oslo Norway

Holz, E.; Kath,O. (1996) Einsatz von B-ISDN-Netzen im Rahmen einer TINA-Kommunikationsplattform, CAMOUFLAGE/S, Humboldt University Berlin

Holz, E.; Wasowski, M.; Witaszek D. et. al. (1996) INSYDE - The final Methodology Report, ESPRIT P 8641, D1.3

Kath, O.(1996) TINA-C und B-ISDN-Netze, Humboldt-Univ. Berlin

Kolberg, M. (1997) Compiler Frontend for TINA-ODL, Humboldt-Univ. Berlin

Niedrig, S.; Rademann, S. (1996) Eine Methodik für den ODP-Konformitätstest und deren Anwendung auf den ODP-Trader, Humboldt-Univ. Berlin

ISO/IEC (1996) DIS Open Distributed Processing - Interface Definition Language

ITU-T (1995) Rec. X.901 | ISO/IEC 10746-1: RM-ODP Part 1: Overview

ITU-T (1995) Rec. X.902 | ISO/IEC 10746-2: RM-ODP Part 2: Descriptive Model

ITU-T (1995) Rec. X.903 | ISO/IEC 10746-3: RM-ODP Part 3: Prescriptive Model

ITU-T (1995) Rec. X.904 | ISO/IEC 10746-4: RM-ODP Part 4: Architectural Semantics

ITU-T (1995) Rec. X.950 | ISO/IEC 13235:RM-ODP ODP Trading Function

SITE (1997) http://www.informatik.hu-berlin.de/Themen/SITE/SDL-tools.html

TINA-C (1994) Overall Concepts and Principles of TINA

TINA-C (1996) Object Definition Language - Manual Version 2.3