COMBINING DESIGN METHODS FOR SERVICE DEVELOPMENT

Marc Born, Andreas Hoffmann,
Mang Li, Ina Schieferdecker
GMD FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin
Tel. +49 30 3463 7 235, Fax. +49 30 3463 8 200
email: {born | a.hoffmann | m.li | schieferdecker} @fokus.gmd.de

Introduction

The Reference Model for Open Distributed Processing (RM-ODP) [5] describes an architecture for the design of distributed services where the basic idea is to split the design concerns into several viewpoints. This is in order to overcome the immense complexity of today's distributed systems by structuring the design process.

Each of the ODP-viewpoints covers different aspects of the system which is to be designed. Though the RM-ODP itself does not define a concrete design methodology, there is a lot of ongoing work concerning this topic. One popular example is the Unified Modelling Language (UML) [3], which provides a set of graphical notations for different design tasks.

This paper presents an integrated approach not only covering the field of service design but also validation and testing of services and reusable service components. It proposes a methodology providing notations and usage guidelines to cover all stages during the development lifecycle of an arbitrary (telecommunication) service.

This methodology is not to be understood as a top-down approach, but it is more an iterative application of each of the stages from an abstract level down to a detailed specification and implementation. Repetition of steps is needed if errors are detected either by validation on the design phase or by testing the implementation. Since the testing step normally takes a lot of the overall development time, an approach to reduce this time via automation of test execution in a distributed Common Object Request Broker Architecture (CORBA) [10] environment is introduced in this paper, too.

The following figure shows the relation between the service development lifecycle and the viewpoint based methodology. As to be seen, there is no fixed one to one relation between the viewpoints and the phases of the lifecycle. Most viewpoints cover topics belonging to different phases.

P. Ciancarini et al. (eds.), *Formal Methods for Open Object-Based Distributed Systems*

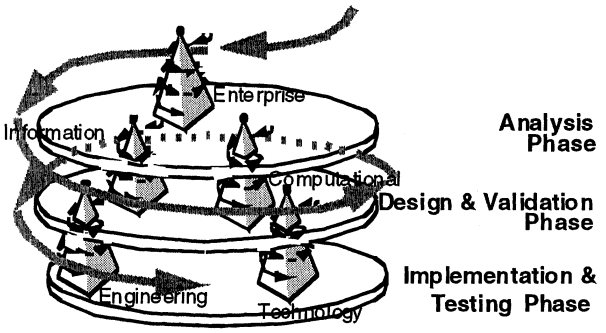


Figure 1: The Overall Development Lifecycle

To describe the computational viewpoint of a service a combination of Object Definition Language (ODL) [9] and Specification and Description Language (SDL) [7] is proposed. Since ODL describes the structure of the service and the signatures only, additional information especially on the behaviour and the connection between the component instances is needed in order to allow validation and automated testing. The component behaviour description should be an abstract one since in most cases only the external visible behaviour should be specified without prescribing any implementation details. In this paper it is shown, that SDL is a good candidate language to do that. It should be noted, that the intention is not to require a detailed SDL model for each individual component. It depends on the application for which component validation and automated testing should be performed. Only for them a SDL behaviour description has to be provided.

The overall methodology is mostly tool supported and has been applied to different projects dealing with the development of telecommunication services.

The paper is structured as follows: The analysis phase section addresses briefly, how the requirements for a new service design task can be captured. Afterwards, it is explained how different modelling languages can be combined to provide a computational model for service components. To increase the quality of the service a validation stage is included into the design phase. In the testing phase section, a method to automate the process of testing an implementation in a CORBA based target environment is described. The SDL based component behaviour description is used for that purpose. Finally, a tool chain supporting the overall method is presented. A summary concludes this paper.

Analysis Phase - Requirement Capturing

The task of capturing the requirements for a new service is not a main issue of this paper. However, it is not trivial and the result of requirement capturing plays an important role during the validation stage.

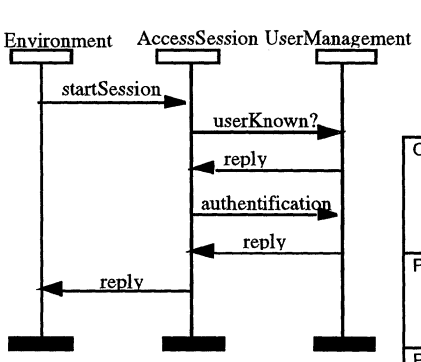
Generally it can be distinguished between functional and non-functional requirements. Examples for non-functional requirements are time constraints or special quality of service requirements like bandwidth. An example for a functional requirement is an expectation on a particular sequence of actions.

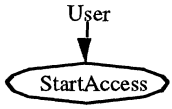
In order to capture the functional requirements, the proposed methodology follows the Use Case approach of Jacobsen which is also supported by the UML. A Use Case contains an abstract sequence of actions which is provided by the system to a certain user in a particular role. Following the enterprise viewpoint of ODP, we describe the actions in terms of policies. Each policy is either an obligation, a permission or a prohibition on the expected system behaviour.

However, the main lack of the Use Case model is that Use Cases are only described informally by plain text. That means, they cannot serve as a basis for automated validation. To overcome this drawback, Use Cases are formalised here by attaching Message Sequence Charts (MSC) [8] on a high level of abstraction to them.

An example to illustrate this is the Access Session taken from the Service Architecture specification of the TINA-Consortium (TINA-C, Telecommunication Information Networking Architecture). It contains the description of a method to get uniform access to telecommunication services from different providers and retailers. This is a very practice related example. Due to the increasing amount of telecommunication services and their providers it is necessary to fix the interfaces between the consumers, the retailers and the providers of services. It is important, that this fixing does not only include signatures but also behaviour. Figure 2 contains one Use Case and its MSC. To start an Access Session the User must be known to the system and his authentication must be successfully.

The difference between the shown enterprise-MSCs and MSCs used for the computational viewpoint is, that the latter include the detailed component structure and messaging information. Enterprise-MSCs are more abstract in this topic.





Obligations	The action is only available if there is an registration for the actor and the actor has authentified itself to the system.
Permissions	The actor has the opportunity to start an Access Session which is than associated to it.
Prohibitions	none.

Figure 2: Formalisation of Use Cases with MSCs

Service Design and Validation Phase

Following the general ODP oriented development process provided in the introduction (see Figure 1) this section discusses at first the information and computational modelling approaches and furthermore the service framework methodology.

The Information Model

The intention for having an information model in ODP is to provide a specification of the kind of information, which is processed by the system and of the relations between the information entities. Additionally, the information model can cover the state information and the state changing information of the whole system by defining static and dynamic schemes.

A well known notation for that purpose is the OMT object model notation, which is also part of the UML. In the tools introduced later on the OMT notation is integrated in order to be able to specify an information model.

The Computational Model

In the Computational Viewpoint, the distributed service is described as a collection of interacting data processing entities, called computational objects (CO). The several COs interacting through their well defined computational interfaces. The service designer focuses on how a particular functionality can be provided without taking into account what kind of computing or network infrastructure is used, when the service is going to be implemented.

Hence, the task of a computational model is to define the object (or component) structure together with the interface signatures on the one hand and to describe the behaviour provided at the interfaces on a high level of abstraction on the other hand.

The computational model plays an important role, because it is the basis for a validation of the system behaviour against the requirements without implementing it on top of a real computing and networking architecture. However, it depends on the used description technique, how abstract the behaviour specification can be kept and whether or not the model can be analysed or validated.

In all cases this viewpoint oriented approach makes error detection easier because it concentrates on errors on the computational level and abstracts from problems with the concrete computing or networking architecture. So, the errors resulting from design mistakes could be detected in an early design stage. It should be noted, that sometimes an abstract model of the service environment has to be provided and included in the computational model of the service.

We use a combination of ODL and SDL to describe the computational model. Object Definition Language (ODL) is a suitable notation to cover the task of defining the structures and signatures of the system. ODL was initially developed by the TINA-consortium. Currently, there is a new question (Q.2) of the International Telecommunication Union (ITU) - Study Group 10 where an ITU Object Definition Language is going to be standardised.

Within the Access Session example the user communicates by the means of an appropriate user application via the provider agent with the user agent located in the retailer domain. The interface i_RetailerNamedAccess provided by the user agent has been chosen for the presented example and is shortened to four operations.

```
module Ret_RP {
    interface I_RetailerNamedAccess{
                    void setUserCtxt (...)              // for the purpose of initialization
                            raises (e_AccessError,e_UserCtxtError);
                    void endAccessSession( ...)         // to close connection to the provider
                            raises (e_AccessError,e_EndAccessSessionError);
                    void startService (...)             // to establish a service
                            raises (e_AccessError,e_ServiceError,e_PropertyError);
                    void endSession (...)               // to terminate a service
                            raises (e_AccessError, e_SessionError);
                    ...
    };


    object UA{
                            supports I_RetailerNamedAccess;
    };
    };
```

These operations are used to do initialization (setUserCtxt), to establish a TINA service (startService), to terminate a TINA service (endSession) and to close the connection to a service provider (endAccessSession). The detailed description of the operation's behaviour is not of interest in the context of this paper and therefore it is not explained here. For reasons of simplicity the following specification example lacks the data type, exception and operation parameter definitions.

With the combination of ODL and a behaviour definition language it is possible to provide the environment with information on how the component behaves. This behaviour description should be abstract enough to ensure that concrete implementation details are hidden to the public when a vendor sells a component. However, even an abstract behaviour description would allow to validate and test the component together with the environment into which it is to be embedded (see testing phase section).

The ITU language SDL is chosen in our methodology to serve as a notation for certain aspects of behaviour description. SDL allows to specify sequences of interactions at the interfaces using the extended finite state machine concept. An SDL specification serves also as a basis for validation, simulation and test case generation. Via simulation MSCs can be generated and validated against (possibly hand written) MSCs from the enterprise or computational viewpoint.

One often hear the argument, that a combination of ODL and a behaviour description technique like SDL is not useful since the information covered by the ODL specification can be described using the behaviour description technique (especially in SDL) directly. However, there are good reasons for applying both languages:

- SDL does not allow to specify IDL data types and signatures. This information is needed for the implementation on top of a CORBA based execution environment,
- the structuring mechanisms in SDL are not intuitive enough and the model becomes often confusing,
- not all components will be fully described with SDL, instead they will be implemented directly. The ODL specification serves as a basis for both ways.

- computational specifications are often prescribed in terms of interfaces which have to be used, for instance TINA reference points or CORBA services. These specifications are given in ODL/IDL and have to be included in the design process.

In the following paragraphs, the methodology for the design of the computational model is explained and shown in Figure 3.

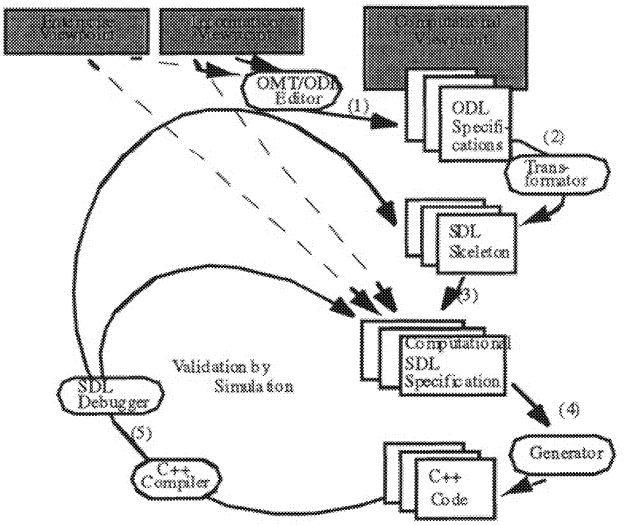


Figure 3: Computational Modelling Process Using SDL

The most important steps of the methodology are:

Step (1): Using the information and the enterprise specification, partially developed in OMT and Use Cases, a computational ODL specification has to be defined.

Step (2): The ODL specification is mapped into a structural equivalent SDL specification.

Step (3): The SDL inheritance feature is applied to enrich the SDL specification generated by step 2 with behaviour descriptions for both the interface and object templates. The behaviour description is based on the specification of states and transitions in dependence of a required support of sequences of operations at the object interfaces and defined exceptions. Step 3 can be repeated to achieve different levels of abstraction in the design of object composition and behaviour. The result is an executable computational model which is not the engineering solution.

Step (4): With help of tool packages the SDL specification from the last step can be checked for correctness of syntax and static semantics. Additionally, it is possible to generate C++ code that can be linked with a simulation library. This leads to a simulator for the SDL system which represents the ODL specification and includes the computational behaviour aspect and hence allows one to check the dynamics of the system.

Step (5): A simulation of the computational model is used to detect design errors prior to implementation. The components of the SDL simulator can be distributed using CORBA. An SDL debugger is a component which supports that kind of validation. Another way of error detection is to explore the state space of the SDL model to find lifelocks or deadlocks. If design errors are detected, a repetition of the steps 1 to 4 could be necessary.

The result of applying this method is a computational model of either a complete telecommunication service or a single component or a set of components.

A part of the behaviour description for the Access Session example is shown in Figure 4.

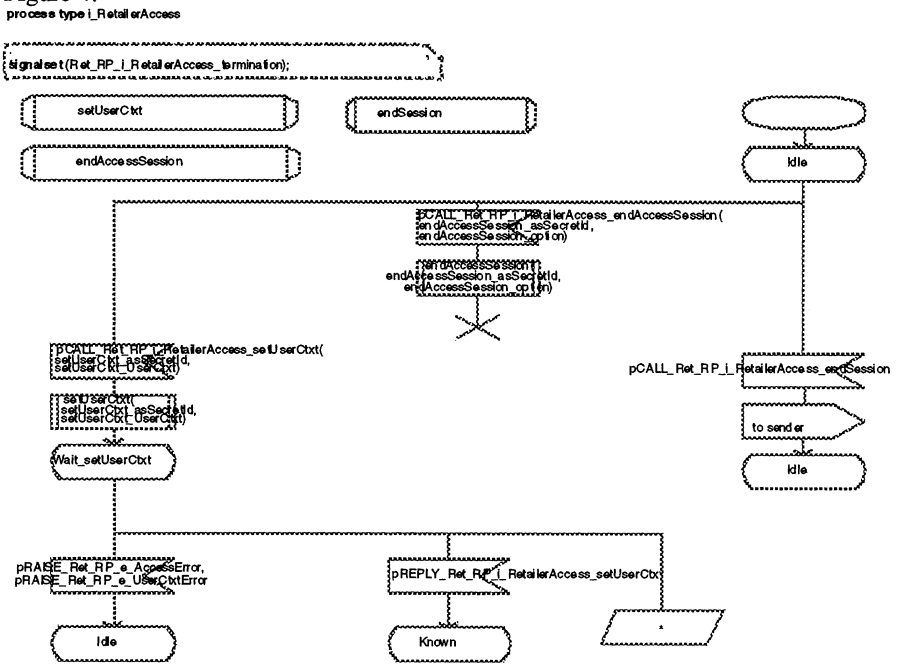


Figure 4: Example of a Behaviour Description in SDL

It is a state machine, describing the behaviour at the i_RetailerAccess interface. In the initial state Idle, there are 2 methods which are allowed: endAccessSession and setUserCtxt. If one of these invocations is received, the appropriate procedure is called. The method endSession is not allowed in this state, therefore, and exception signal is generated and the procedure is not called. This approach allows to model exceptions even in SDL'92 which is essential for specifying telecommunication services.

Testing Phase - Quality Ensurance

Once the implementation is ready for execution, it has to be assured that it behaves like expected, i.e. it is conform to the SDL model. This is of special importance if a service is composed of components provided by different vendors or downloaded from the network.

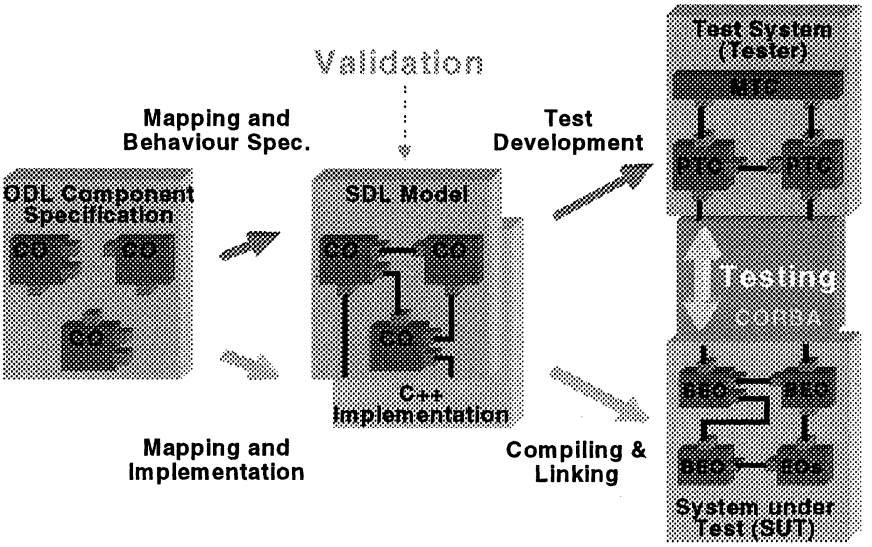


Figure 5: Test Case Development and Execution

As shown in Figure 5 the structure of the service specified in ODL is mapped to both the implementation as well as the SDL model of it. Since the ODL specification does not contain any behaviour description (only textually), it has to be added to the implementation and the SDL model separately. Even if some of the implementations of the components are generated automatically from the SDL specification, testing is needed because errors might be contained in the code generator or target environment. Hence, it is not assured that the implementation follows the SDL specification. To bridge this gap executable test cases have to be developed from the validated SDL specification. To reduce the effort an automation of the testing process is needed to make it efficient and repeatable and to make test results comparable.

It is not sufficient to validate the services on specification level with a formal description technique like SDL only, but also to test them in the target environment. Testing is a general method to determine that the implementation behaves like the specification, i.e. the implementation is conform to the specification. It may be used to check

- service components individually,
- conformance to standardised reference points or protocols,

- the individual service components working together as well as
- to check individual services working together in a multi-service environment.

The widely used and standardised notation for conformance testing is TTCN (Tree and Tabular Combined Notation) [12]. It should be noted that conformance testing can only show the presence of functional errors and cannot guarantee their absence. Hence, conformance testing is not a means to prove a 100% compliance to a specification.

Telecommunication services are distributed systems and may be tested by means of a centralised or distributed test system. Since the complexity of a centralised tester magnitudes larger than a distributed tester for our approach a distributed test system has been chosen. Such a test system can be structured similar to the system under test itself: each PCO (point of control and observation) per interface of a service component is controlled and monitored by a parallel test component (PTC). A main test component (MTC) is used to co-ordinate them.

Test Case Development

The starting point of any automated testing is the design and specification of test cases, which are used to check certain conformance requirements. Following our methodology test cases are derived from the computational SDL specification (semi-automatically or manually).

To enable automatic test case derivation a test purpose description on the basis of MSCs has to be provided as an input for the TTCN tool. It should be mentioned that the MSCs generated by an SDL simulator/validator during the validation process might be used for this purpose. The TTCN tool can than try to derive the appropriate test case behaviour from the SDL specification following the test purpose described by the MSC.

For semi-automatic step-by-step derivation of a test case the developer has to select all signals to be sent as test requests to the SUT. The TTCN tool can than derive all possible reply signals from the SDL specification. In both automated cases the constraints part of a TTCN test suite as well as test suite operations have to be added manually.

The implementation of TTCN-based tests can be supported by TTCN compilers that translate abstract test case description into program code according to the TTCN semantics. The TTCN/CORBA gateway supports the bridging between TTCN-based executable test cases and CORBA ORBs as the execution environment. The gateway is the general test access to CORBA-based systems. Further details of TTCN-based test description and the TTCN/CORBA gateway can be found in [13].

Summary of Methodology and Tool Support

In order to get acceptance for new software development methods it is not enough to propose the method from a theoretical perspective only. Tools have to be provided to enable the usage of the method. In this section we will show how already existing tool packages and new developed tools can be combined and integrated to support the whole lifecycle of service development as introduced in the introduction.

Analysis Phase

GMD FOKUS has developed a tool called Y.SCE [6] which provides graphical notations for the Use Case Model and an ODP based organisational model. High level MSCs can be captured using the TAU tool set from Telelogic. Both tools will be connected to allow that MSCs can be attached to the Use Cases and a navigation form one tool to the other is possible.

Design & Validation Phase

The Y.SCE tool provides a graphical notation for ODL and is able to import and export textual ODL and CORBA-IDL files.

The Y.SCE tool is able to generate SDL files which afterwards can be imported into SDT from Telelogic. Navigation is again possible. The behaviour information can be added using SDT. This tool provides also the functionality of simulation and validation of an SDL specification and the ability to check an SDL specification against the requirement specification of the enterprise model partially provided by enterprise-MSCs.

Implementation & Testing Phase

To support the implementation of a service, there are different tools which can generate implementation language code from a SDL specification. This code can be enriched manually, if the SDL model is not fully complete (i.e. generic).

For the development of TTCN test cases the ITEX tool set supplies powerful means. Since both ITEX and SDT are integrated into the TAU tool set provided by Telelogic the derivation of test cases from an SDL specification is well-supported. Especially TTCNLink and AutoLink should be mentioned for semi-automated or automated derivation of test cases.

To bridge the gap between the abstract test suites and executable ones GMD FOKUS has developed a tool set to make abstract test suites executable in an distributed CORBA environment. The ETS generator generates code from the TTCN test suite which can be executed by means of the TTCN-CORBA gateway to test an implementation (SUT) automatically.

Conclusion

ODP provides an architecture for the description and development of distributed systems by the use of different viewpoints, each of them highlights specific aspects of a distributed system.

This paper proposes a methodology to improve the development process for telecommunication services by an integrated use of selected formal description techniques. The presented methodology is based on applying well-established modelling techniques such as IDL/ODL, SDL and TTCN to the development process of telecommunication services. It addresses mainly the design & validation phase and the implementation & testing phase.

The methodology uses IDL/ODL specifications to describe interfaces of and relations between computational objects. Automated support is provided to derive SDL skeletons from the IDL/ODL specifications, which are in a next step enhanced

with behaviour model to address dynamic behavioural aspects of the telecommunication service. The completed SDL specifications of central and/or critical components, subsystems and/or the complete service are the basis for validation. Validation via simulation and automated state space exploration is a means to check certain requirements and properties of the telecommunication service to be built.

To ensure that the delivered service behaves like the customer expects, failures in the implementation should be detected before shipping. This can not only be achieved by validating the SDL based design model but by testing the real implementation. For that reason another part of the presented methodology is the (semi)automated derivation of test cases, which are described in TTCN, and the automated test case execution, which is supported by the presented test code generator and the TTCN/CORBA gateway. Once the telecommunication service has been implemented, properties of the telecommunication service in the target environment can be checked by applying these test cases and assigning test verdicts.

Future work will address the integration of modelling techniques for the enterprise and information viewpoint and will consider the further development of the tool-based framework specialisation and implementation.

References

[1] Born, M.; Fischer, J.; Winkler, M.: Formal Language Mapping from CORBA IDL to SDL'92 in Combination with ASN.1, Informatik-Bericht Nr. 44, Humboldt-University Berlin, 1995

[2] Fischer J., Fischbeck N., Born M., Hoffmann A., Winkler M.: Towards a behavioural Description of ODL, TINA'97 conference

[3] G. Booch, I. Jacobson, J. Rumbaugh: Unified Modelling Language 1.1, OMG 1998.

[4] Farley, P.; Hogg, S.; Kristiansen, L. et al.: Ret Reference Point Specifications, Snapshot 1, Version 0.4, TINA-Consortium, May 14, 1997

[5] ITU-T Rec. X.903/X.904 | ISO/IEC 10746-3/-4: 1995, Open Distributed Processing - Reference Model Part 3/4.

[6] GMD Fokus Berlin: Y.SCE Tool, User Manual, 1997.

[7] CCITT/ITU-T: SDL - Specification Description Language, Z.100, Genf,1992.

[8] CCITT/ITU-T: Message Sequence Charts, Z.120, Genf, 1996.

[9] TINA-C Baseline Document: TINA Object Definition Language Manual, Version 2.3. July 1996.

[10] OMG: The Common Object Request Broker Architecture and Specification, Version 2.1. Aug. 1997.

[11] Telelogic: The Tau 3.2 Tool set Guidelines, Telelogic, Malmö, Oct., 1997

[12] ISO/IEC 9646: Part 3: Information technology - Open Systems Interconnection - Conformance testing methodology and framework. The Tree and Tabular Combined Notation (TTCN), Edition 2, Nov. 1997.

[13] A Hoffmann, M. Li, I. Schieferdecker: Conformance Testing of TINA Service Components - the TTCN/CORBA Gateway - , IS&N 98, Antwerp, 1998.