International Conference on Communication Technology
ICCT'98 October 22-24, 1998 Beijing, China

Applying a Framework Approach with Validation to the Design of Telecommunication Services

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

Abstract:
Due to the highly increasing complexity of new telecommunication services and the distributed nature of them on the one hand and the requirement to come up with a short time to market on the other hand, new methods, techniques and tools covering the whole service creation process are needed.
This paper presents an integrated approach covering the fields of designing, validating and testing services and reusable service components. For the design stage a methodology which applies concepts from the Reference Model for Open Distributed Processing (ODP) is introduced.

To ensure that the service to be designed meets the requirements of a potential user, a validation stage has been included into the design methodology. This paper also contains a new method for automated testing of distributed services through executing these test cases in a CORBA based target environment to check whether or not the implementation fulfills the specification.

1. Introduction

In order to overcome the immense complexity of current telecommunication systems the Reference Model for Open Distributed Processing (RM-ODP) [10] [11] describes an architecture for the design of such services where the basic idea is to split the design concerns into several viewpoints. Though the RM-ODP itself does not define a concrete design methodology, there is a lot of ongoing work concerning this topic.

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.

The process 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 plane 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) [16] environment is introduced in this paper, too.

The following figure shows the relation between the service development lifecycle and the viewpoint based methodology.

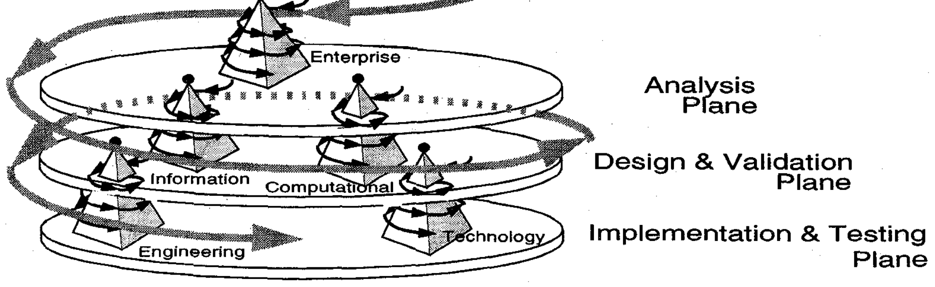

Figure 1: The Overall Development Lifecycle

Since the time to market must be as short as possible, it is not realistic to start for each new service from the scratch. Instead, as many things (components) as possible have to be taken from already existing services and systems and have to be adopted to provide the new functionality.

This leads to the idea of building flexible software frameworks and to use them on the design and implementation planes. The main idea is to describe a framework as a set of components with well-defined configurations. The parts of the behavior which are specific to the concrete service are left open and called flexibility points or holes. A special kind of tools called Paradigm tools provide different views on the holes and allow to fill them with predefined or generated behavior. If no specific behavior is available, the framework has to be modified in order to cover the new requirements for the service to be designed.

To describe the components contained in a framework and their relations a combination of Object Definition Language (ODL) and Structure and Description Language (SDL) [13] is proposed.

Since ODL describes the structure of the service and the signatures only, additional information especially on the behavior and the connection between the component instances is needed in order to allow validation and automated testing. The component behavior description should be an abstract one since in most cases only the external visible behavior 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 has to be noted, that the intention is not to require an 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 behavior 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.

2. Analysis Plane - Requirement Capturing

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

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. An 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 behavior.

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 formalized here by attaching Message Sequence Charts (MSC) [15] on an high level of abstraction to them. The difference between those enterprise-MSCs and MSCs used for the computational viewpoint is, that the later include the detailed component structure and messaging information. Enterprise-MSCs are more abstract in this topic.

3. Service Design and Validation Plane - Applying the Framework Approach

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.

3.1. 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. However, the practical experience shows, that the information model is often omitted depending on the kind of service which is to be produced. That is why the focus is more on the computational model. It should be noted, that information entities can also be described using the techniques provided for the computational viewpoint. For example datatype definitions in ODL can be used for information modelling.

3.2. 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.

The computational model plays an important role, because it allows to validate the system behavior against the requirements without implementing it on top of a real computing and networking architecture. This 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. Additionally, ODL has the power to serve as an abstract component language because the objects and object groups have well defined interfaces which describe the information needed by the environment to integrate the object or object group (component) into a given set of existing components. With an ODL specification the potential dependencies between components are defined in terms of supported and required interfaces.

Additionally, with the combination of ODL and a behavior definition language it is possible to provide the environment with information on how the component behaves. This behavior 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 behavior description would allow to validate and test the component together with the environment into which it is to be embedded (see Section 4).

The ITU language SDL is chosen in our methodology to serve as a notation for certain aspects of behavior 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.

In the following paragraphs, the methodology for the design of the computational model is explained and shown in Figure 2.

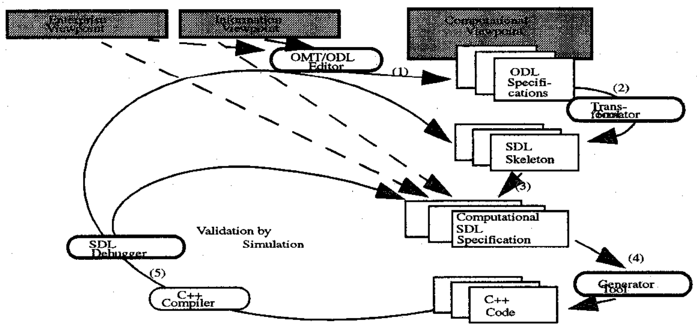

Figure 2: Computational modelling process using SDL

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. This can be done automatically and is founded by a language mapping which is published in [3].

Step (3): The SDL inheritance feature is applied to enrich the SDL specification generated by step 2 with behavior descriptions for both the interface and object templates.

Step 3 can be repeated to achieve different levels of abstraction in the design of object composition and behavior. The result is an executable computational model which is not the engineering solution. It defines only a behavior model to observe and evaluate special aspects of object behavior which are visible from outside. Especially complex object configurations can be studied.

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.

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.

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. The reusability of these components is supported by a so called computational database or repository (see Section 5 for details on tool support). In the following section, a framework approach is presented where the component representations are developed using the methodology presented above.

3.3. Building service frameworks

To enable rapid service creation, the TOSCA project has adopted and extended the framework idea known from object-oriented technology. A framework is "almost ready" service software and provides reusable classes of computational objects (service components) and well-defined configurations for them. It consists of a fixed part for the service class, and a flexible part, which contains flexibility points or holes. Flexibility points are place holders in the descriptions of computational objects for things like parameters, types, operations, and behavior. Constraints can be used to restrict the set of possible replacements. Flexibility points have to be filled during the service creation process to get a complete model of the service.

The framework contains three types of components:

- Fixed components which must not to be changed,
- Generic components with have some minor flexibility points and may have default behavior for the holes,
- Abstract components with an completely open behavior.

It is important to note that a framework is much more than just a reusable component library. It can be seen as an architecture where new service functionality can be supplied by using the functionality provided by the framework. The framework does not only contain the reusable components but it also covers the relations between them and the method which a user must apply to make use of the provided functionality and to adopt it to fulfill its requirements.

The framework (i.e. its components) itself can be built by applying the methodology described in the previous section.

4. Implementation & Testing Plane - Quality Ensurance with Automated Testing

After the instantiation of the framework it has to be assured that the implementation behaves like expected, i.e. like the SDL model. This is of special importance if a service is composed of components provided by different vendors or downloaded from the network. As shown in Figure 3 this problem can only be solved by testing.

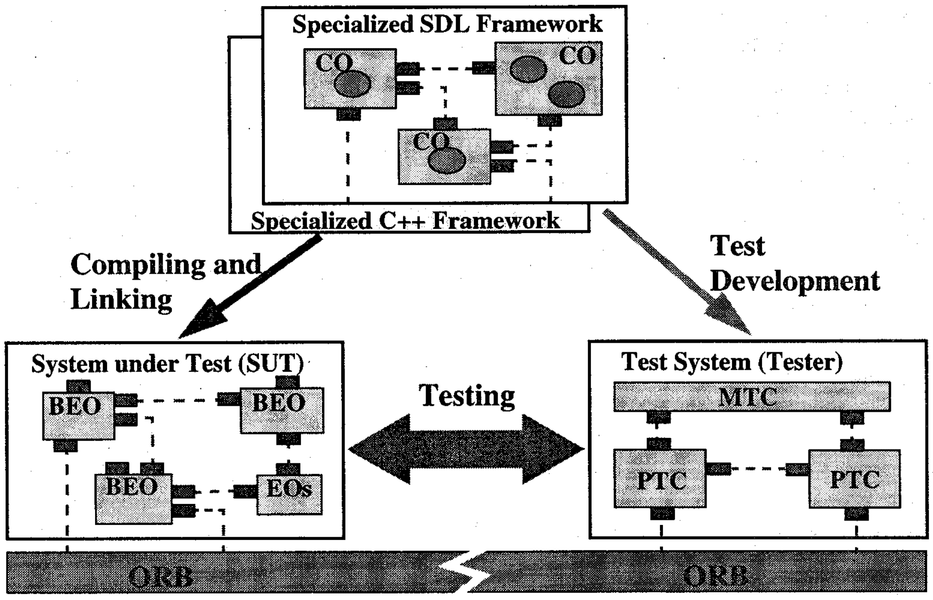

Figure 3: Testing of the instantiated Framework

4.1. Conformance Testing in the Overall Approach for Service Creation

As stated above, due to the increasing complexity of new telecommunication services, the need to be able to validate and test large and heterogeneous telecommunication systems is becoming increasingly important. In particular, 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 standardized 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 standardized notation for conformance testing is TTCN (Tree and Tabular Combined Notation) [21]. It should be noted that conformance testing can only show the presence of functional errors and cannot guarantee their absence. Consequently, conformance testing is not a means to prove a 100% compliance to a 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.

Telecommunication services are distributed systems and may be tested by means of a centralized or distributed test system. Since the complexity of a centralized 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. A main test component (MTC) is used to coordinate them (see Figure 3).

4.2. Objectives for Testing Services and Service Components

Conformance testing considers the service, i.e. the system under test (SUT), to be a black-box, i.e. the internal structure and the internal behavior is invisible. The test system interacts with the system under test only via the system's interfaces and assigns test verdicts after comparing observed reactions from the system under test with the expected ones.

For component testing the objective might be to test the individual computational objects that are used for realizing service components. A component of a service is considered to be a black-box, so that groups of computational objects cannot be distinguished from single computational objects. In particular, each interface of the service component is tested separately.

From the ODP perspective, testing of computational objects is based on the structuring information of service components. It uses therefore a grey-box testing approach. The service components are not considered to be a black-box, but rather their internal structure, i.e. the configuration of computational objects with their interfaces has to be known. Due to political reasons, it might be practically useful for in-house tests rather than for third-party tests at an external test lab. However, testing the individual computational objects can use the same testing technique like testing service components or services - it is the question which implementation parts are accessible and observable by the test system.

Validation and testing of components proofs that components work correctly in separation. However, it should be noted that it does not assure that a composition of components which have been validated and tested separately behaves like expected. Component validation and testing can only minimize the risk that a configuration of components does not work together in a way expected.

4.3. Test Execution - The TTCN/CORBA Gateway

To enable the execution of the developed test suites in a distributed CORBA environment a gateway is needed. This is because the implementation of abstract TTCN test cases for service components deals with the integration of a message-oriented system into an object-oriented system.

Internally, the Gateway consists of a configurative client part and a server part, and a main part responsible for initialization and management of the involved Gateway components. The client part is the requester on services provided by the SUT objects and is therefore the active test part. The server part emulates behavior of the server objects that may be invoked by the SUT objects. To support conversion of messages into CORBA compliant operation handling and vice versa, the CORBA DII (Dynamic Invocation Interface), DSI (Dynamic Skeleton Interface) and the Interface Repository are applied.

Finally, it should be emphasized that the implementation of the gateway described above is CORBA compliant and supports features of Concurrent TTCN such as parallel test components and their management.

5. 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 Section 1.

Analysis Plane

GMD FOKUS has developed a tool called Y.SCE [12] which provides graphical notations for the Use Case Model and an ODP based organizational model. The information is stored in a repository (database). High level MSCs can be captured using the TAU toolset 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 Plane

The Y.SCE tool provides a graphical notation for ODL and is able to import and export textual ODL and CORBA-IDL files. The ODL information is stored in the same repository as the Use Case information. It is possible to establish links between the Use Cases and the ODL model and to navigate along these links.
The Y.SCE tool is able to generate SDL files which afterwards can be imported into SDT from Telelogic. Navigation is again possible. The behavior 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.

In order to enable the framework specialization both tools interact with the paradigm tools being under development in the context of the TOSCA project.

Implementation & Testing Plane

Following the paradigm-tool supported and framework-based service creation process the implementation of a service will be done by specialization of the framework through a paradigm-tool. This tool is currently under development by the project TOSCA. The specialized framework is ready for compilation.

For the development of TTCN test cases the ITEX toolset supplies powerful means. Since both ITEX and SDT are integrated into the TAU toolset 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 toolset 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.

6. 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. Distributed telecommunication services that for example follow the TINA architecture are a specific kind of ODP systems. They are highly distributed and complex and as such hard to develop.

This paper proposes a methodology to improve the development process for telecommunication services by an integrated use of selected formal description techniques on the one hand and by applying service frameworks to enable rapid service creation on the other hand. 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 plane and the implementation & testing plane.

The presented methodology emphasizes on the computational and engineering viewpoint modelling. Modelling techniques for the enterprise and information viewpoints such as MSCs and UML are covered but not yet fully integrated into the methodology.

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 behavior model to address dynamic behavioral 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.

The methodology presented can be applied to many different kinds of software that is to be produced, but it becomes much more powerful in combination with the service framework approach. The major advantage of this integrated approach is that the time to market can be reduced dramatically. The idea is, to reuse as much components as possible from existing services to be able to react in a short time to new requirements.

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 paradigm-based framework specialization and implementation.

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

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

[4]. G. Booch, I. Jacobson, J. Rumbaugh: Unified Modeling Language 1.1

[5]. v.Löwis, M.: Using CORBA in an SDL Simulation Environment. DOCT Frankfurt, 1997.

[6]. OMG: CORBA Component Model RFP, orbos/97-06-12, 1997

[7]. DSTC, IONA: Revised Submission on Multiple Interfaces and Composition, orbos/97-01-02, OMG, 1997

[8]. Project TINA-Platform (Distributed, Integrated Platforms for Multimedia Telecommunications, Management, Mobility and Security) is commissioned by DeTeBerkom, a subsidiary of Deutsche Telekom Group, http://www.fokus.gmd.de/ovma/general/projects/tangram/

[9]. Project TOSCA (TINA Open Service Creation Architecture): ACTS (Advanced Communications Technologies and Services) Project

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

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

[12]. GMD Fokus Berlin: Y.SCE Tool, 1997

[13]. CCITT: SDL - Specification Description Language, International Standard Recommendation Z.100, Genf,1992

[14]. CCITT: SDL in combination with ASN.1, International Standard Recommendation Z.105, Genf, 1996.

[15]. CCITT: Message Sequence Charts, International Standard Recommendation Z.120, Genf, 1996

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

[17]. A. Schill: DCE - das OSF Distributed Computing Environment, Einführung und Grundlagen. Springer Verlag, 1996.

[18]. Telelogic: The Tau 3.2 Toolset Guidelines, Telelogic, Malmö, Oct., 1997

[19]. J. Grabowski: SDL and MSC Based Test Case Generation - An Overall View of the SAMSTAG Method. - Technical Report IAM-94-005, University of Bern, May 1994.

[20]. ISO/IEC 9646: Information technology - Open Systems Interconnection - Conformance testing methodology and framework. 1994.

[21]. ISO/IEC 9646: Part 3: The Tree and Tabular Combined Notation (TTCN), Edition 2, Nov. 1997.

[22]. A Hoffmann, M. Li, I. Schieferdecker:Conformance Testing of TINA Service Components - the TTCN/ CORBA Gateway - ,accepted paper for IS&N 98.