The ITU-ODL to C++ Mapping and its Integration into an SDL based Design-Methodology for Telecommunication Services

Marc Born, Andreas Hoffmann, Mario Winkler
GMD FOKUS Kaiserin-Augusta-Allee 31, D - 10589 Berlin, Germany
Tel. +49 30 3463 7 235, Fax. +49 30 3463 8 200
email: {born | a.hoffmann | winkler}@fokus.gmd.de

1. Introduction and Motivation

The Object Definition Language (ODL) [6] is a language initially designed by the Telecommunication Information Networking Architecture (TINA) consortium to develop specifications of TINA systems for the Open Distributed processing (ODP) computational viewpoint [7]. However, the audience for ODL as a description technique goes far beyond the TINA community. Advanced concepts of TINA-ODL like multiple interfaces, groups and stream interfaces influence the development of other definition languages. The Object Management Group (OMG) [10] is on the way to include multiple interfaces into OMG-Interface Definition Language (IDL) and has recognized the importance of stream flows by requesting a stream control interface language binding. The international standardization body ITU-T has established a new question (SG10/Q.2) to develop an ITU-ODL based on TINA-ODL.
Since ITU-ODL is suitable to describe the structure of the service and their signatures only, additional information especially on the behavior and the connection between the component instances is needed. A description of behavior is the basis to enable early validation of the specification and automated testing of the implementation.

However, in most cases the behavior description of the service components should be abstract since only the external visible behavior should be specified without prescribing any implementation details. In several papers [1] it has been shown, that SDL [9] is a good candidate language to do that. SDL provides a formal semantic which allows validation and there are tools which generate implementations for an SDL specification. It is also possible to derive TTCN [11] tests cases semi-automatically from SDL. A summary of a design methodology which combines the use of ITU-ODL and SDL is contained in Section 2.

However, it is not realistic 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 an SDL behavior description has to be provided. As a consequence the usual case is, that not all component implementations can be derived from their SDL specification via automatic code generation. Especially if performance aspects have to be considered or if the concepts of SDL are not suitable to specify the component behavior in any detail the components have to be implemented by hand.

In this case there is a need to have language mapping not only from ITU-ODL to SDL but also to an implementation language like C++ in order to use the structural information contained in the ITU-ODL specification for the implementation.

Since ITU-ODL is a strict superset of OMG-IDL all aspects concerning the communication between the components of a system are described using OMG-IDL interface descriptions. In order to be able to use existing ORB implementations as a platform to implement an ITU-ODL specification, the mapping for the OMG-IDL part of ITU-ODL is adopted. The intention is, that the existing ORB specific OMG-IDL compilers can be used to map the IDL part of an ITU-ODL specification to C++.

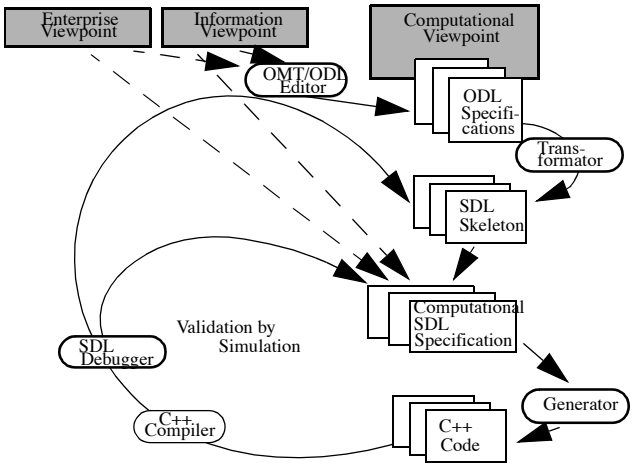
Additionally, the structural information for computational objects or components should be reflected in the implementation language as well. This information reflects design decisions made during system design. It is not a

requirement to map it into the implementation language, since the application will work even if only the IDL part is mapped (Only the interfaces are of importance for communication). But ITU-ODL should be understood as a computational design language and the structural information contained in it makes the programming of distributed applications easier. Therefore, a language mapping of ITU-ODL to C++ is proposed in this document.

2. Overall Design Methodology

The usage of ITU-ODL and SDL is integrated in a methodology covering the whole development lifecycle of a distributed telecommunication service. This methodology is based on ODP. It is a complete description of both, stages and steps involved in the analysis, design and implementation phase. The methodology enables the integrated usage of different languages and techniques like Use Cases, OMT [2], ITU-ODL, SDL and TTCN by defining mappings and/or relations between the different notations.

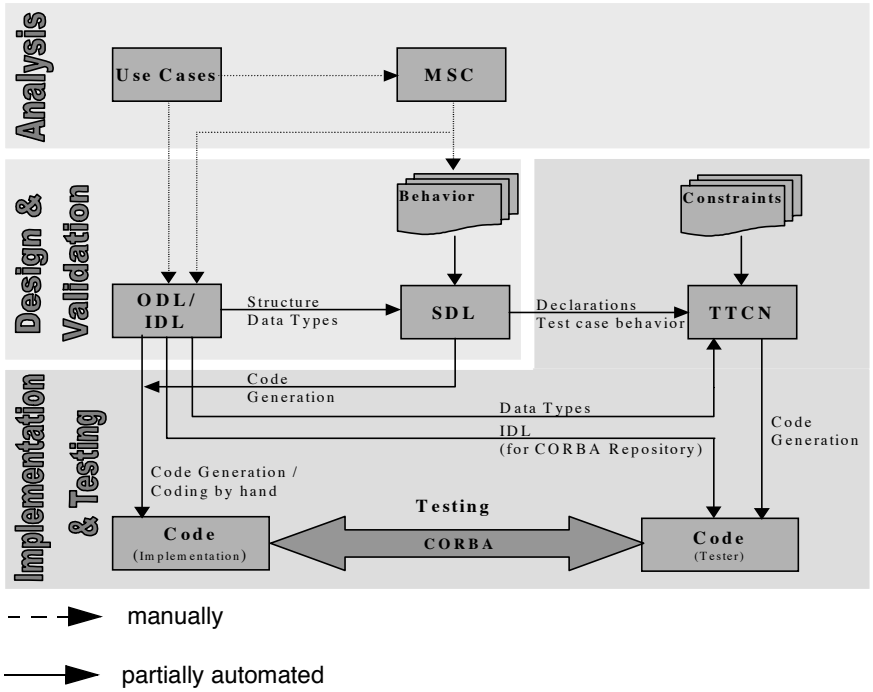
The process is not to be understood as a top-down approach, but it is more an iterative application of each of the stages from an abstract level down to a detailed specification and implementation. Repetition of steps is needed if errors are detected either by validation on the design phase or by testing the implementation. The following figure gives an overview on the methodology and shows the integrated usage of ITU-ODL and SDL.



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

However, as motivated in the previous section, it is not realistic to describe the complete behavior for each individual component with SDL. In most cases it is sufficient to specify the externally visible behavior at the interfaces of the components. Afterwards test cases can be derived and an implementation can be tested whether it is conform to the behavior specification or not.

For that reason, another way of making use of ITU-ODL is needed. That is to map ITU-ODL directly into the implementation language. Such an implementation language mapping is shown in the next section on the example of C++. The advantage of that mapping is, that the computational design information is reflected in the generated implementation skeleton. Additionally the mapping of the interfaces ensures that the implementation is able to communicate with its environment in a well-defined way. If test cases and testers are derived from the SDL specification, they can run against the implementation via these interfaces. The following figure shows the used notations and the information flow between them. The language mapping rules from ITU-ODL to C++ are described in the next section.



3. ITU-ODL to C++ Mapping Rules

Since ITU-ODL is a superset of OMG-IDL the language mappings contained in CORBA serve as guideline to define a language mapping for ITU-ODL. One goal is to map the IDL parts of ITU-ODL as defined there to be able to use the IDL compilers delivered with existing CORBA products. This will increase the expectance of the proposed mapping since a lot of current distributed platforms are CORBA-based.

However, from a general point of view the mapping rules are independent from CORBA and can be used for several kinds of Distributed Processing Environments (DPE). In the following sections, the mapping for only those concepts which are not part of IDL is explained.

3.1. Flows (Stream Interfaces)

ITU-ODl stream interface definitions consists of a set of data flow definitions. Such dataflows have a well-defined frame type and an attribute specifying the direction of the flow, i.e. whether the interface serves as a sink or source for the specific data flow.

There is no proposal for a mapping of flows currently. A possible solution might be to adopt the results from the OMG in the area of AV-streams.

3.2. Mapping for Computational Object Templates

ITU-ODL CO definitions consists of a set of interface templates which are offered by the described object to its environment, a set of interface templates which ate needed by the CO to perform its service and an interface template, which is instantiated at the time where the CO itself is instantiated. This initial interface serves as a handle to the CO. It may contain operations to obtain other interfaces, to perform identity checks etc.

Computational object templates are mapped to C++ classes (CO classes). The main functionality which is provided by the generated classes is, that they contain methods for creation and deletion of the supported interfaces of the ITU-ODL CO definition.

Additional functionality such as checkpointing or migration can be provided but is not prescribed and depends on the platform on which the application should run. For that reason it might be useful to have a common base class similar to ::CORBA::Object, and let all generated classes inherit from it.

Required Interface Specification

Required interface templates are not mapped themselves. There is the CORBA client mapping for the interface template only.

However, the required interface information is needed, if a minimal server implementation (with respect to the number of generated classes) should be built. In that case only the client part of the CORBA mapping for those interfaces which are required and the server part for the interfaces which are supported has to be generated.

Supported Interface Specification

The interface templates announced in the list of supported interfaces on an arbitrary CO are mapped in the following way:

- There is a C++ class generated which implements the class generated according to the OMG-IDL mapping. This class contains implementations for all operations of the interface. The behavior is to delegate an incoming call to an implementation class.
  The programmer does not have to touch any of these delegation classes.
- The implementation class is generated for all interfaces listed somewhere as supported. The implementation class contains all operations as pure virtual methods. The programmer is responsible to provide implementations of these implementation classes.
- There is a possibility to make a reference to an instance of an implementation class known to the delegation class instance. This can be implemented either in the constructor of the delegation class or by a method __set_implementation.

The CO can create instances of its supported interface templates in the following way:

1. Create an instance of the delegation class.
2. Obtain a reference to an instance of the implementation class (can exist or be created).
3. Make the reference to the implementation class instance known to the delegation class instance.

Initialization Specification

The initial interface is mapped in the same way as supported interfaces. It is task of the used DPE to provide an instance of the initial interface to the instantiater of the CO. The way how to do this is not prescribed by this mapping.

Inheritance

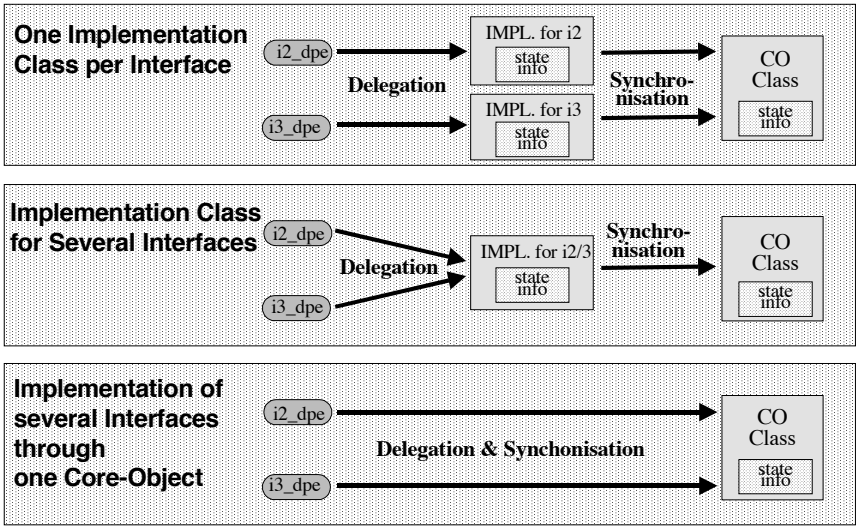
Object template inheritance is realized as C++ inheritance between the generated C++ classes.

The programmer is free in the choice how to structure the application. He must only provide implementations for the implementation classes and possibly for the CO classes.

The following figure shows some examples how the implementation might be structured. It is assumed that there is one computational object which supports the interfaces i1, i2 and i3.

- There is the possibility to have one implementation for each interface implementation class. That normally happens when there is a lot of state information related to the interface. The synchronization and information exchange is handled via the CO class implementation.
- It is possible to have one implementation for a number of interface implementation classes.
- If there is no state information related to the interfaces, the implementation could be done directly in the CO

class implementation.



3.3. Mapping for Object Group Templates

ITU-ODL Object group templates are sets of computational objects and computational groups which can be clustered together for any reason. This concept has been adopted from ODP. It is a structuring concept for a wide range of applications. The reason for grouping them is contained in the group predicate specification. Since the predicate specification can vary, there is no prescribed mapping for object groups. Examples for a group predicate can be that all members of the group belong to the same domain or that they provide together a particular service.

However, if the purpose of the group is to group its members for implementation reasons, the group can be mapped onto a class definition in the same way as COs. The contracts of the group (the supported and required interfaces) are than mapped in the same way as for computational objects.

4. Conclusion

A design methodology for distributed systems can normally not be based on a single notation, since each notation is suitable to cover specific aspects of the system only. The trend goes towards an integrated application of different notations and techniques. SDL plays an important role in the design methodology mentioned in this paper since it is the basis for early validation and automated testing. The structures and signatures of the system are defined using the language ITU-ODL. Since not all components have a behavior description in SDL or this behavior description could not serve for automatic code generation it is necessary to have a different approach to come to an implementation. Therefore, a language mapping from ITU-ODL to C++ is introduced in this paper. This language mapping allows flexible implementation structuring and ensures, that the developer can make use of the computational design information in the implementation stage. The mapping rules are aligned with those from ITU-ODL to SDL and from SDL to TTCN. This ensures the application of automated testing facilities and increases the benefits resulting from the application of ODL and SDL in the design methodology.

The methodology introduced in this paper including the mapping from ITU-ODL to C++ is tool supported.

Future work will be on the mapping for stream concepts and object groups.

5. References

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

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

[3]. OMG: Control and Management of A/V Streams RFP (Telecom RFP1), telecom/96-08-01, OMG 1996

[4]. OMG: Multiple Interfaces and Composition RFP , orb/96-01-04, OMG 1996

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

[6]. TINA-C: TINA Object Definition Language MANUAL, Version 2.3, 1997

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

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

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

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

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