

support is a must for the specification and development of large systems. Currently a series of companies have started to develop tools for UML, supporting not only the specification development, but also code generation and reengineering. Especially the last two topics are of increasing importance. The pure generation of pure code skeletons is not sufficient, code generation for the behavior parts is required too (and supported by non-UML tools used in the telecommunications industry). Different ways for the combined usage of SDL and UML have been identified. For the next time a complete definition of translation rules and meta-model extensions is planned. Tool support is a crucial point for the successful application of this technology. However, the IDL specification of UML is a good starting point for the specification of mapping rules and the tool development in a vendor independent way.

5 REFERENCES

- [1] OMG: "The Common Object Request Broker Architecture CORBA 2.1", OMG, 1997
- [2] ITU-T Rec. X.901 | ISO/IEC 10746-1: „Open Distributed Processing - Reference Model Part 1: Overview“, ITU-T '95
- [3] ISO/IEC CD 14750, „Open Distributed Processing - Interface Definition Language“, ISO/ITU-T DIS '96
- [4] ITU-T Rec. Z.100: „SDL ITU-T Specification and Description Language“, ITU-T, 1992
- [5] Rational: „UML Semantics V1.1“, 1997
- [6] Rational: „UML Notation Guide V1.1“, 1997
- [7] TINA-C „Overall Concepts and Principles of TINA“, TINA-C 1994
- [8] TINA-C: „Object Definition Language - Manual Version 2.3“, TINA-C, 1996
- [9] Olsen, A.: „Introduction of Gate Types in SDL“, ITU-T Temporary Document, Tele Danmark, 1997
- [10] Holz, E.: „ODP Architectural Semantics in SDL“, ISO JTC1 SC21 Input Document, 1996
- [11] E. Holz: „Application of UML within the Scope of new Telecommunication Architectures“, GROOM Workshop on UML, Mannheim, Physica-Verlag, 1997
- [12] Holz, E.; Wasowski, M.; Witaszek et.al.: „INSYDE Methodology“, ESPRIT Project 8641, 1996.
- [13] Telelogic: „The SDT User Manual“, Telelogic, 1996.
- [14] Gou, F.; MacKenzie, T.: „Translation of OMT to SDL-92“, Proc. of SDL'95, Oslo, 1995.
- [15] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen: Object Oriented Modeling and Design. Prentice Hall, International Editions (1991).

3 REFERENCE MODELS FOR DISTRIBUTED SYSTEMS

The management of the large amount of information usually involved in the complete specification of a complex distributed system is addressed by the concept of viewpoints. Viewpoints are separations of concern. They give a partial view of the complete system specification. Five different viewpoints are identified in the RM-ODP [2]:

- Information viewpoint
- Enterprise viewpoint
- Computational viewpoint
- Engineering viewpoint
- Technology viewpoint.

Each viewpoint is associated with a viewpoint language expressing the rules which are relevant to the respective universe of discourse.

To verify the conformance between real implementations and specifications and the compliance between two specifications the RM-ODP introduces the concept of reference points. Reference points are selected interaction points serving as potential conformance points. Four different classes of reference points have been introduced:

- Programmatic Reference Points
- Perceptual Reference Points
- Interworking Reference Point
- Interchange Reference Point

Additional information required when testing an implementation of an ODP specification is called Implementation eXtra Information for Testing (IXIT). Those information relate Implementation Conformance Statements (ICS) - i.e conformance points, required behavior etc. - to their realization in an implementation. In order to test the conformance of an implementation under test the tester has to provide sets of IXIT's and ICS relating the concepts and structures of the enterprise, information and computational specification of the system to its engineering specification and a set of IXIT's and ICS relating the concepts and structures of the engineering specification to the implementation choices made in the technology specification/implementation.

The RM-ODP defines concepts and common functions of distributed systems in a generic manner. Refinements and specializations of this model are explicitly encouraged and are currently under way (e.g. TINA-C), other standard bodies have set up relations between their model and the RM-ODP (e.g. OMG). Standards for the realization of common functions, so called component standards, fill out the framework and are providing a base for the development of ODP applications.

The development of viewpoint specifications is a typical application area for different specification and description languages. For the information viewpoint as well as the enterprise viewpoint OOAD techniques like UML found extensive usage, whereas for the computational viewpoint a combination of formal techniques and interface/object definition languages is applied. The possibility to use SDL and UML in combination will ease the transition between specifications of different viewpoints. The planned changes to SDL (exception handling, object-oriented data types, gate types) will even increase the ability of SDL to be applied as computational language and to relate such SDL specifications to UML specifications of the information/enterprise viewpoint.

4 CONCLUSION

Formal description techniques and object oriented analysis and design techniques are two important instruments for the development of complex and distributed software systems. Whereas FDT's have been developed for the specification of unambiguous system descriptions and for the verification and validation, the focus of OOAD techniques is an efficient design process from early analysis stages down to the implementation. The question should be therefore not whether to use either FDT or OOAD, but how to combine the advantages of both techniques. Even more in the advent of new and upcoming description techniques (e.g. OMG-IDL, TINA-ODL) and with the need to specify systems from different viewpoints (ODP, [2]), ways to combine different techniques and to make statements about the relations between specifications of different viewpoints become increasingly important.

The extension features of UML (e.g. stereotypes) make it possible to tailor the language to the application area, gaining more compact specifications. This could be supported by libraries/packages of predefined concepts. Tool

- sage flow find a correspondence in MSC's.
- Parameterized Classes (templates)
This concept is best reflected by SDL types with context parameters, although restrictions on the type of context parameters are more stringent.
- Packages
The UML package has its direct correspondence in SDL packages. The import or use dependencies are mapped to packages use-clauses.

Design process rules

In this approach, which puts parts of the transition rules in the design process, far-reaching use of the UML extension features (stereotypes, properties) is made. The general idea here is to provide a repository of modeling concepts which directly reflect SDL concepts (extended meta-model). This repository or package serves as the initial package for the development of the UML model, i.e. an UML model to be translated to SDL has to use thew concepts of this repository. An example for the definition of the concept BlockType is given in the next figure:

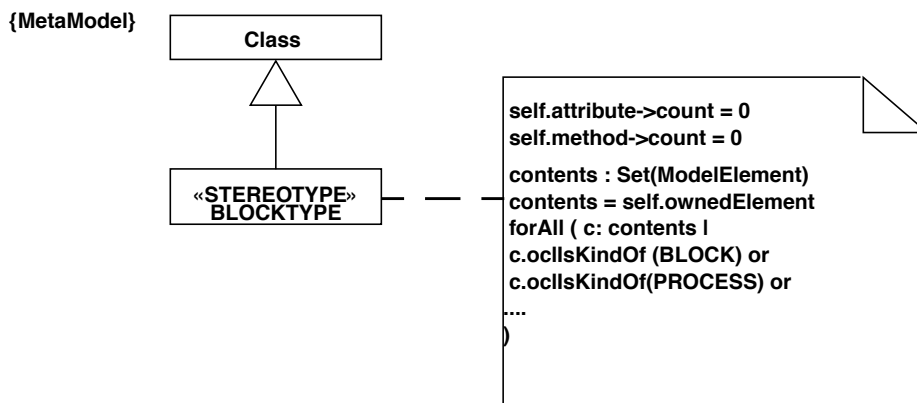


Figure 5: Definition of Blocktype stereotype

If applied for a concrete UML specification, the translation from UML to SDL happens as follows:

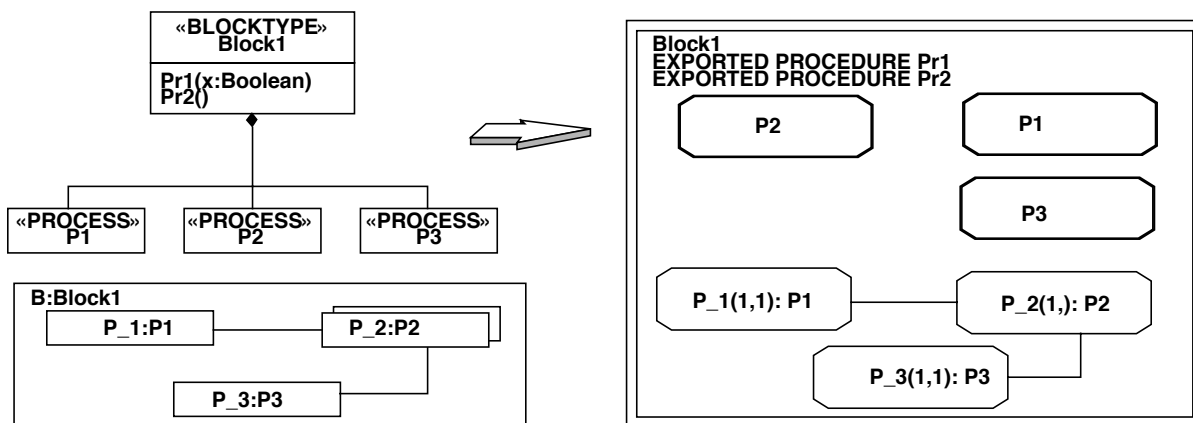


Figure 6: Application of stereotypes

Although the targets for mapping of single UML constructs are usually the same as in the general translation approach, the translation process itself can be made simpler. This is caused by the additional structure and configuration requirements implied by the stereotypes and the properties of the extended meta-model. Moreover, the extended meta-model can also serve as repository of modeling constructs for the modeling of SDL specifications with UML.

other techniques (UML sequence charts to MSC). An example for the transformation of an UML static diagram is given in the next figure.

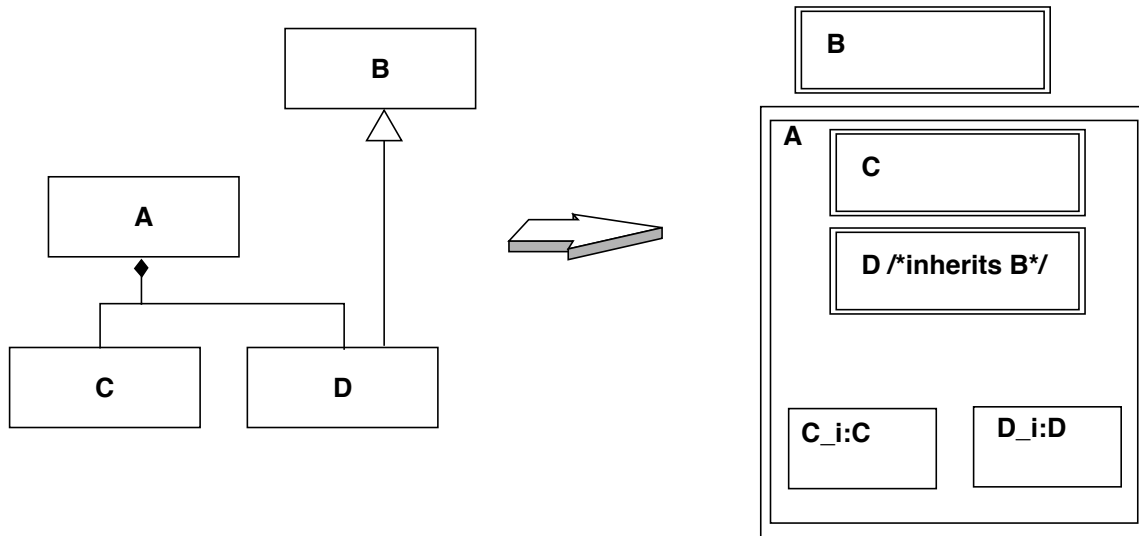


Figure 3: Mapping of static structures

A new feature of UML for the description of behavior are activity charts. A translation of such an UML activity charts to SDL process/service state-transition graphs is shown in the next figure. Such a translation can be made

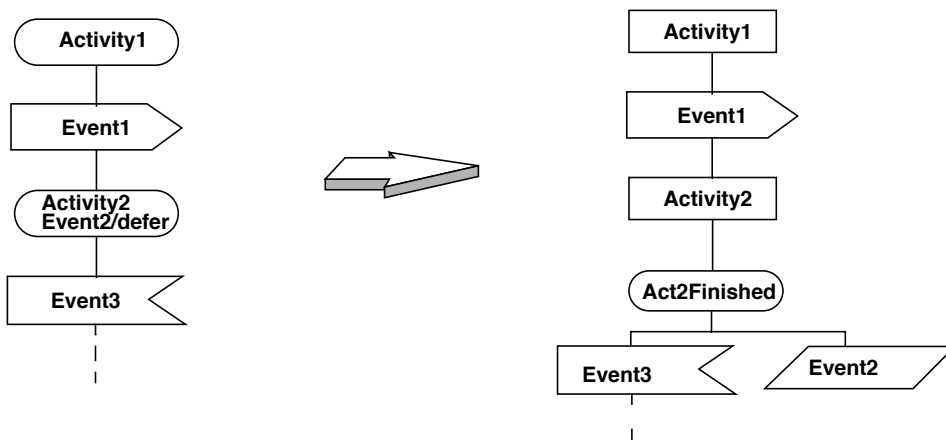


Figure 4: Translation of UML activity diagrams

rather straightforward. Proposals for the translation of some further new concepts are listed below:

- Interfaces
Interfaces can not be directly reflected in SDL due to the lack of a corresponding construct. However, the fact that a class implements an interface can be reflected by attaching a gate with operations as defined in the interface to the SDL construct onto which the class is mapped. Additionally, a set of remote procedure definitions can be derived from the interface.
The current proposal for gate types in SDL would provide a direct target for the UML interface and could simplify a translation.
- Collaborations
Collaborations as a set of objects related in a particular context are mapped to SDL block and/or process diagrams. Links between the objects are reflected by channels resp. signalroutes transmitting the messages specified in the collaboration diagram. The sequencing information are not directly reflected, however, interactions/mes-

The modeling approach as proposed here uses UML only to visualize information, which are contained in an SDL specification, but not explicitly visible in its SDL/GR representation. A way back from UML to SADL is not foreseen, the specifier only works with SDL. This makes the development of tools rather straight-forward, the first step is the information collection (type definitions, relations) and the second step the construction of the UML model. For the second step many commercial UML tools provide an appropriate API to build and visualize such a model under control of an external tool.

2.2 Combined Specification with SDL and UML

A combined specification with UML and SDL has many similarities with the OMT-SDL approaches. This is of course not surprising due to the fact that OMT was one of the most influential sources for the UML development. However, UML is much less ambiguous as OMT, which makes the transition to SDL less restrictive and more flexible. The main idea again is to use UML in the analysis and early design phases and to apply SDL in later design stages down till implementation. Such an approach does also allow mixed technologies in the design phase, e.g. SDL for parts of the system and CORBA IDL/C++ for other parts of the system.

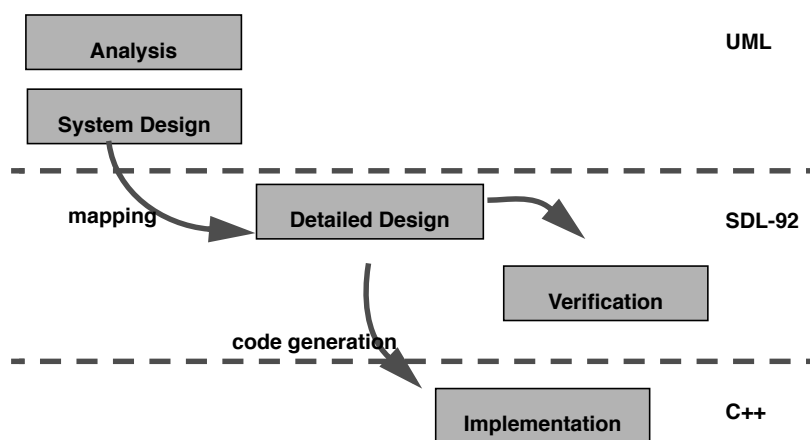


Figure 2: Design steps

The main task in such a development process is a well defined transition from UML to SDL (and possibly other techniques with clearly defined communication interfaces between the subsystems). In general two ways to go from UML to SDL can be identified:

- put rules in translation process
no extensions/restrictions on UML notation, but possibly incomplete translation
- put rules in design process
stereotypes and properties to reflect SDL concepts
reverse mapping.

Translation process rules

In this case most rules which have been defined for OMT could be applied for the corresponding UML submodels (static structure and state chart diagrams). The principal idea here is to map UML classes to corresponding SDL type definitions (system, block, process or service types). These rules can be - compared to the OMT rules - even more simplified because the UML semantics is more strict and clear than the OMT semantics. New rules have to be defined for those submodels which do not have a predecessor in OMT (e.g. collaborations, activity charts etc.), although not all models will find a direct reflection in the SDL specification (e.g. use cases) are will be related to

itance relationships. Second, type definitions may have the same name, if they belong to different kinds of types or occur at different scope units and they may be nested. The first problem can be solved by special UML stereotypes (blocktype, servicetype,...) enhancing the UML meta model (cf. Figure 5). A solution to solve the naming problem is to use a full SDL pathname for an UML class, although this would make the model less readable. The problem of nested types can only partly be solved by dependencies.

The table lists the SDL concepts and their corresponding UML representation.

SDL	UML	Comment
<type definition>	Class	a stereotype should be used to differentiate between system, block, process and service types
<typebased definition> <system definition> <block definition> <process definition> <service definition>	Object	type based definitions are part of type definitions or of simple definitions; the corresponding UML Objects will occur as an endpoint in a composition aggregation
type definitions with formal context parameters	Template	
specialization	Generalization relation	
nested type definitions	Dependencies	UML does not have nested class definitions, however, the dependency-concept can partly reflect this semantic relation
channels, signalroutes	Associations	

An example for an UML static diagram of an SDL specification is given in the next figure.

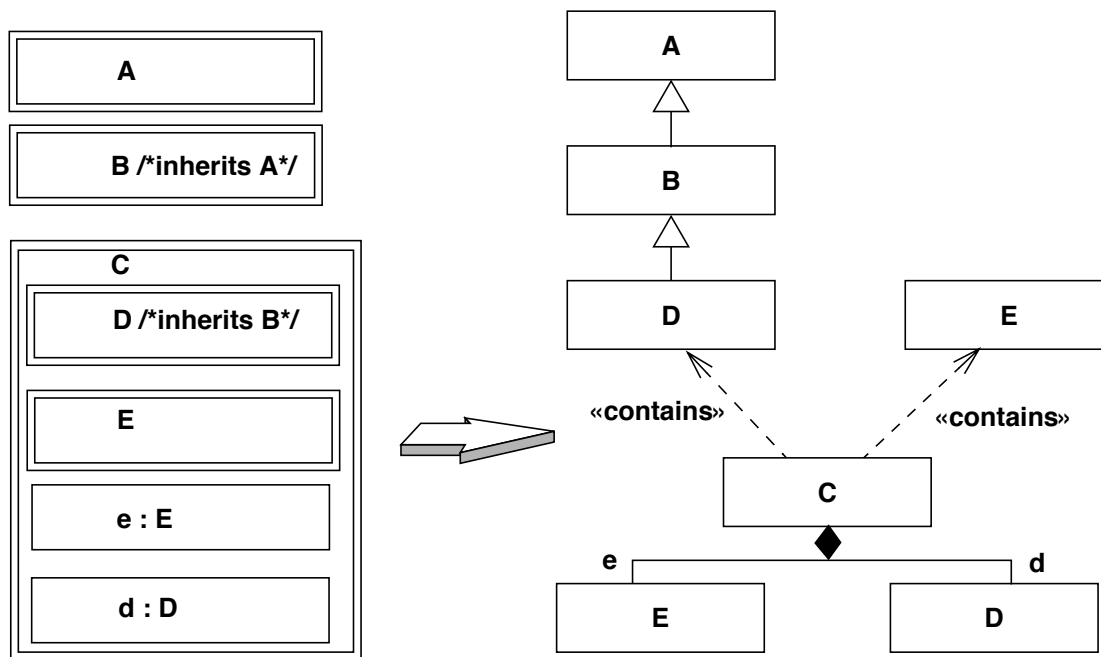


Figure 1: UML model of an SDL specification

cation consists of a configuration of blocks and processes (according to the OMT class structure) and the behavior descriptions for the processes (process graphs) derived from the OMT state machines. The resulting SDL specification, which will be more or less complete - depending on the OMT description, can now be analyzed with the tools and methodologies available for SDL. Finally the verified and completed specification serves as a starting point for the code generation. Unfortunately, the code generated from such an SDL specification bears less object orientation as code generated directly from a complete OMT description. The second approach uses both techniques at the same level of the development process [13]. The application of OMT is restricted to the specification of data types, whereas the overall system structure and behavior is specified in SDL. The connection between both specifications is made by so-called links. This enables the use of object oriented data types (defined in OMT) within a SDL specification.

2 RELATING SDL AND UML

The standardization of UML V1.1 as well as the new standardization activities for SDL (SDL-96, exception handling concepts, extended Remote-Procedure-Call- concept, object-oriented data types, gate types) provide a base to revise the approaches to combine SDL with OOAD techniques. Two main directions for the joint application of SDL and UML can be identified:

- Modeling SDL specifications with UML
- Application of UML in the analysis and early design phases and SDL in the later design stages.

The first direction serves mainly the idea to make large SDL specification better understandable and to give additional information (e.g. inheritance hierarchies, dependencies, pattern structures) for documentation purposes or as additional implementation advice. The second direction follows mainly the translation approach as it has been developed for OMT.

2.1 Modeling SDL specifications with UML

With the introduction of object-oriented concepts in SDL only minor changes have been made to the graphical representation. The main additions are:

- special symbols for types (block, process, service, system type)
- special symbols for communication connections (gates)
- special symbols for existing type based elements (block, process, service).

All other features are - even in the graphical representation - expressed by text constructs. Dependencies between type definitions and between type definitions and type based definitions are not explicitly visible, what can make hard to discover the results of changes to the design and may lead even to errors. It is therefore helpful to have a graphical representation of these implicit relations and dependencies.

The UML static structure diagrams are a well suited notation for this purpose. In detail these diagrams are among other topics able to

- show the classes and objects of a system including their attributes and operations,
- show associations between the classes,
- show inheritance/generalization relations between the classes and
- dependencies between the classes.

These features are sufficient enough to visualize the information missing in a graphical SDL specification. The necessary information to construct such an UML diagram can be directly derived from the textual version of the SDL specification, what helps to automate this task. To avoid over or double specification, behavior concepts (state-transition graphs) are not reflected in the UML diagrams. Also in most cases, the class definition will just consist of the name compartment, whereas the attribute and operation compartments will be suppressed. However, the operation and attribute compartment could be used to list the complete set of signals, remote procedures and remote variables, which the specified unit can receive resp. supports. To structure the specification UML packages can be used in parallel to SDL packages.

Two major problems with this approach are related to the differences in naming and scoping in SDL and in UML. First, SDL does have different kinds of types (system, block, process, service), which will not be mixed in inher-

Application of UML in the SDL Design Process

Eckhardt Holz

Humboldt-Universität zu Berlin, Dept. of Computer Science

A.-Springer-Str. 54a, 10117 Berlin, Germany, Phone +49-30-20181 241,

Fax +49-30-20181-234, e-mail holz@informatik.hu-berlin.de

Abstract

Formal specification techniques as SDL and Object oriented analysis and design techniques as UML have found widespread application in the design of distributed systems. However, in many cases a initial decision has been made to either use an FDT or to use an OOAD technique. The paper proposes a more integrated solution, combining the advantages of SDL with the advantages of UML. Different approaches for the combination of both techniques are introduced and compared. Finally an outlook on the influence of reference models and standards for distributed systems on the selection and combination of different design languages and techniques is given.

Keywords

Design Process, Object-Orientation, UML, SDL, Formal Specification

1 INTRODUCTION

Object oriented analysis and design techniques (OOAD) have been applied successfully for the development of large applications. Their advantage is that the object oriented paradigm is consistently applied throughout all stages of the design process up to the implementation. With the definition of the Unified Modeling Language (UML, [5], [6]) finally also a broadly accepted common notation technique has been found. It is expected that UML, pushed by the standardization of UML by the OMG, will find an increasing importance for the design and implementation of large and distributed systems. On the other side, within the telecommunication domain traditionally formal specification and description techniques as SDL, MSC or Estelle play an important role. These techniques are not only used for the notation of standards but are also applied in the design process. Tools support this with a variety of features, from graphical editors over analyzers to code generators.

An other important fact which can be observed within the last years is the fading of the differences between distributed systems in the computer industry domain and systems from the telecommunications domain. Although coming from different roots, developers in both domains increasingly target similar problems. This will also influence methodologies, notations and tools applied within the design process. The question for the future should however not be OOAD/UML instead of formal techniques but rather how techniques like SDL and UML can be used in fruitful combination.

Within different projects proposals for a combination of the OOAD and UML predecessor technology OMT [15] and the FDT SDL [4] have been made. Two main directions can be identified there:

- Translation of OMT to SDL
- Link between SDL and OMT models.

The translation approach ([12], [14]) uses OMT and SDL at different stages within the development process: OMT is applied for the analysis and early system design, SDL is applied for the detailed design and as a high-level implementation language. A translational semantics for OMT is defined by giving rules to map the different concepts of OMT to SDL concepts. Such a mapping can be supported by tools. The main base for the translation are the OMT class model and the behavior model, the functional model is not considered. The resulting SDL specifi-