

UML Profile for Communicating Systems

*A New UML Profile for the Specification and Description of
Internet Communication and Signaling Protocols*

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

vorgelegt von
Constantin Werner
aus Salzgitter-Bad

Göttingen 2006

D7
Referent: Prof. Dr. Dieter Hogrefe
Korreferent: Prof. Dr. Jens Grabowski
Tag der mündlichen Prüfung: 30.10.2006

Abstract

This thesis presents a new Unified Modeling Language 2 (UML) profile for communicating systems. It is developed for the unambiguous, executable specification and description of communication and signaling protocols for the Internet. This profile allows to analyze, simulate and validate a communication protocol specification in the UML before its implementation. This profile is driven by the experience and intelligibility of the Specification and Description Language (SDL) for telecommunication protocol engineering. However, as shown in this thesis, SDL is not optimally suited for specifying communication protocols for the Internet due to their diverse nature. Therefore, this profile features new high-level language concepts rendering the specification and description of Internet protocols more intuitively while abstracting from concrete implementation issues. Due to its support of several concrete notations, this profile is designed to work with a number of UML compliant modeling tools. In contrast to other proposals, this profile binds the informal UML semantics with many semantic variation points by defining formal constraints for the profile definition and providing a mapping specification to SDL by the Object Constraint Language. In addition, the profile incorporates extension points to enable mappings to many formal description languages including SDL. To demonstrate the usability of the profile, a case study of a concrete Internet signaling protocol is presented. Furthermore, a proof-of-concept implementation for the generation of full behavioral and structural SDL design specifications from UML models has been developed.

Zusammenfassung

Die vorliegende Dissertation beschreibt ein neues Unified Modeling Language 2 (UML) Profil für kommunizierende Systeme. Es ermöglicht die eindeutige und ausführbare Spezifikation und Beschreibung von Kommunikations- und Signalisierungsprotokollen mittels der UML speziell für das Internet. Dadurch können Spezifikationen bereits vor der Implementation analysiert, simuliert und validiert werden. Das Profil basiert auf der gesammelten Erfahrung und den Sprachkonzepten der Specification and Description Language (SDL). Im Gegensatz zu anderen Profildefinitionen benutzt dieses Profil formal beschriebene Einschränkungen zur Definition des Profils und zur Spezifikation von Abbildungsregeln nach SDL. Dies ermöglicht ein automatisiertes Überprüfen auf Korrektheit der Abbildung. Es unterstützt erweiterte Sprachkonzepte, die speziell zur Beschreibung von Internetprotokollen ausgerichtet sind, für die SDL nur eingeschränkt oder aufwändig verwendbar ist. Des Weiteren definiert das Profil spezielle Erweiterungspunkte, um neben SDL noch weitere Abbildungen auf andere formale Beschreibungstechniken zu ermöglichen. Durch die Unterstützung von mehreren konkreten Darstellungsmöglichkeiten eines Modells kann das Profil mit einer Vielzahl von UML 2 kompatiblen Modellierungswerkzeugen eingesetzt werden. Zusätzlich ist eine Implementation entwickelt worden, die es erlaubt, ein vorliegendes UML 2-Modell in eine vollständige textuelle SDL Struktur- und Verhaltensbeschreibung zu übersetzen.

Acknowledgments

I am grateful to my supervisor Prof. Dr. Hogrefe for many fruitful and helpful discussions in the stimulating atmosphere of his research group. I also thank my co-supervisor Prof. Dr. Grabowski for accepting the task of examining this thesis.

Many thanks go out to all my colleagues from the Institute of Informatics at the University of Göttingen. Especially, I am grateful to my colleagues Michael Ebner, Rene Soltwisch and Xiaoming Fu for useful and valuable discussions on formal modeling techniques.

I also thank my students Julia Woch and Sebastian Kraatz for contributing to this work with their considerable efforts. This thesis would not be in its current shape without the comments of numerous people. I thank Omar Alfandi, Mohammed Alfandi, Andrea Hellner, Ingo Juchem, Helmut Neukirchen, Katharina and Peter Samow for reading this thesis, finding errors and providing helpful suggestions for improving this thesis.

Last but most important, I am deeply indebted to my parents, my family and Katrin for their patience and continuous support during the preparation and writing of this thesis. Without them, this thesis would never have been finished.

Table of Contents

ABSTRACT	III
ZUSAMMENFASSUNG	V
ACKNOWLEDGMENTS	VII
TABLE OF CONTENTS	IX
LIST OF FIGURES	XI
1 INTRODUCTION	1
1.1 SCOPE OF THIS THESIS	2
1.2 RELATED WORK	3
1.3 NOVELTY OF THIS PROFILE.....	5
1.4 STRUCTURE OF THIS THESIS	5
2 FUNDAMENTALS OF COMMUNICATION PROTOCOLS	7
2.1 NETWORK LAYERS	7
2.2 COMMUNICATION PROTOCOLS	8
2.3 NETWORK SERVICES	10
2.4 ARCHITECTURES OF COMMUNICATION PROTOCOLS	11
2.5 SUMMARY	14
3 DESCRIPTION TECHNIQUES FOR COMMUNICATION PROTOCOLS	15
3.1 SPECIFICATION OF SERVICES AND PROTOCOLS.....	15
3.2 MODEL-BASED DEVELOPMENT	17
3.3 FORMAL DESCRIPTION TECHNIQUES	19
3.4 THE SPECIFICATION AND DESCRIPTION LANGUAGE	24
3.5 THE MESSAGE SEQUENCE CHARTS	30
3.6 SUMMARY	31
4 THE UNIFIED MODELING LANGUAGE	33
4.1 HISTORY OF UML	33
4.2 THE UML VERSION 2.....	36
4.3 UML AND METAMODELING	42
4.4 UML EXTENSION MECHANISMS	46
4.5 THE OBJECT CONSTRAINT LANGUAGE.....	49
4.6 XML METADATA INTERCHANGE	51
4.7 SUMMARY	52
5 ANALYSIS OF SDL WITH RESPECT TO INTERNET COMMUNICATION PROTOCOLS	53
5.1 CASE STUDY OF A SIGNALING PROTOCOL: RSVP	53
5.2 LANGUAGE CONCEPT FOR THE UML CS PROFILE.....	62
5.3 SUMMARY	68
6 OVERVIEW OF THE UML CS PROFILE	69
6.1 ARCHITECTURE	70
6.2 BEHAVIOR	79
6.3 DATA.....	86
6.4 SUMMARY	87
7 PROFILE DEFINITION	89
7.1 INTRODUCTION.....	89
7.2 STRUCTURE	94

7.3	BEHAVIOR	112
7.4	ACTIVITIES	136
7.5	RANDOM	154
7.6	CONCEPT OF TIME	158
7.7	DATA TYPES	163
7.8	SUMMARY	174
8	SEMANTICS OF THE UML CS PROFILE	177
8.1	TRANSLATIONAL SEMANTICS FOR UML CS PROFILE	177
8.2	OCL-BASED MAPPING TO SDL-2000	183
8.3	EXAMPLE OF AN IMPLEMENTATION: AN XSLT-BASED APPROACH.....	203
8.4	SUMMARY	205
9	CASE STUDY.....	207
9.1	ARCHITECTURE	207
9.2	BEHAVIOR	213
9.3	SUMMARY	220
10	CONCLUSIONS AND OUTLOOK	223
	REFERENCES.....	225
	ABBREVIATIONS	235
	APPENDIX A: UML CS PROFILE IN XMI 2.1	239
	APPENDIX B: XSLT STYLESHEET FOR UML CS.....	277
	APPENDIX C: SDL DIAGRAMS OF THE RSVP MODEL.....	291
	CURRICULUM VITAE.....	303

List of Figures

FIGURE 1: COMPUTER NETWORK ARCHITECTURE DECOMPOSED INTO LAYERS	8
FIGURE 2: COMMUNICATION BETWEEN TWO LAYERS.....	9
FIGURE 3: INTERACTION OF SERVICE USERS AND ENTITIES.....	10
FIGURE 4: THE OSI REFERENCE MODEL.....	12
FIGURE 5: ISO OSI REFERENCE MODEL AND TCP/IP ARCHITECTURE COMPARED.....	13
FIGURE 6: PROTOCOL ENGINEERING PROCESS	15
FIGURE 7: CONCEPT OF A MODEL	18
FIGURE 8: INTERPRETATION OF FORMAL DESCRIPTIONS	19
FIGURE 9: SDL ARCHITECTURAL MODELING ELEMENTS	27
FIGURE 10: AN SDL SYSTEM DEFINITION	27
FIGURE 11: GRAPHICAL NOTATION ELEMENTS IN SDL-2000.....	28
FIGURE 12: SDL PROCESS DESCRIPTION CONSTRUCTS	29
FIGURE 13: SDL CONSTRUCTS FOR COMMUNICATION	30
FIGURE 14: BASIC LANGUAGE CONSTRUCTS IN MSC.....	31
FIGURE 15: FROM METHOD WARS TO THE UML.....	34
FIGURE 16: EVOLUTION OF THE UML.....	35
FIGURE 17: OVERVIEW OF UML 2 DIAGRAM TYPES.....	37
FIGURE 18: UML REPOSITORY AND REPRESENTATIONS.....	37
FIGURE 19: EXAMPLE OF A CLASS DIAGRAM.....	39
FIGURE 20: EXAMPLE OF A COMPOSITE STRUCTURE DIAGRAM.....	39
FIGURE 21: EXAMPLE OF A STATE MACHINE DIAGRAM	41
FIGURE 22: EXAMPLE OF AN ACTIVITY DIAGRAM	42
FIGURE 23: FOUR LAYER METAMODELING ARCHITECTURE	43
FIGURE 24: INSTANCE CREATION IN UML METAMODELING LAYERS.....	43
FIGURE 25: OVERVIEW OF UML 2 METAMODEL CLASS HIERARCHY	45
FIGURE 26: UML EXTENSION MECHANISMS	46
FIGURE 27: GENERALIZATION AND EXTENSION NOTATION IN THE UML	48
FIGURE 28: DATA TYPES AND SPECIALIZED TYPES IN OCL.....	50
FIGURE 29: XMI METAMODEL LAYERS.....	51
FIGURE 30: THE FORMAL PROCESS.....	55
FIGURE 31: RSVP NETWORK SCENARIO MODEL GENERATED IN TAU 4.6	57
FIGURE 32: INTERNAL NETWORK STRUCTURE BLOCK TYPE OF ALL NF NODES	58
FIGURE 33: MESSAGE FLOW OF A RSVP RESOURCE RESERVATION	59
FIGURE 34: INITIAL MESSAGE EXCHANGE AFTER ROUTER SHUTDOWN.....	60
FIGURE 35: NEW ROUTE AND STATE ESTABLISHMENT	61
FIGURE 36: EXCERPT OF THE NEW ROUTE ESTABLISHMENT MESSAGE FLOW USING LOCAL REPAIR.....	61
FIGURE 37: BLOCK INTERCONNECTION IN A MULTIPLE NODE NETWORK USING A SINGLE GATE	66
FIGURE 38: BLOCK INTERCONNECTION IN A MULTIPLE NODE NETWORK USING MULTIPLE GATES	66
FIGURE 39: MODELING OF A UML CS SYSTEM	69
FIGURE 40: UML METACLASS OVERVIEW OF THE AGENT CONCEPT.....	71
FIGURE 41: HIERARCHICAL DECOMPOSITION OF A SYSTEM.....	72
FIGURE 42: LOCAL OPERATION DEFINITION	73
FIGURE 43: SYSTEM WITH BLOCKS	75
FIGURE 44: SIGNAL AND SIGNALLIST DEFINITIONS	76
FIGURE 45: DEFINITION OF CHANNELS	77
FIGURE 46: DYNAMIC PORTS ATTACHED TO BLOCK.....	78
FIGURE 47: GENERALIZATION OF AGENTS.....	79
FIGURE 48: STATE MACHINE WITH ONE STATE.....	80
FIGURE 49: EXCERPT OF TRANSITION METACLASS IN THE UML METAMODEL.....	81
FIGURE 50: ALTERNATIVE NOTATION FOR TRANSITION TRIGGER AND GUARD.....	82
FIGURE 51: DIFFERENT NOTATIONS FOR SIGNAL OUTPUT.....	83

FIGURE 52: ALTERNATIVE BEHAVIOR DESCRIPTION IN ACTION	83
FIGURE 53: TASK BOX	84
FIGURE 54: DECISION NODE WITH CONDITIONS	84
FIGURE 55: MERGE NODE	85
FIGURE 56: TERMINATE AND RETURN NODE	85
FIGURE 57: NOTATION FOR SOFT STATE	86
FIGURE 58: DEFINITION OF PRIMITIVE AND COMPOSITE TYPES	87
FIGURE 59: DECLARATION OF COMPOSITE DATA TYPE AND TYPE SEMANTICS	87
FIGURE 60: SPECIFICATION AND DESCRIPTION PROCESS USING UML CS PROFILE	90
FIGURE 61: EXAMPLE OF STEREO TYPE DEFINITION TABLE	90
FIGURE 62: INTERFACE DEFINITION OF SCHEDULER	93
FIGURE 63: DYNAMIC PORTS AND CHANNEL PATHS	107
FIGURE 64: DECOMPOSITION OF MULTIPLE INSTANCES WITH SINGLE CHANNEL MULTIPLICITY	110
FIGURE 65: DECOMPOSITION OF MULTIPLE INSTANCES WITH CHANNEL MULTIPLICITY OF TWO	110
FIGURE 66: EXTRACT OF THE TRANSITION METACLASS WITH RESPECT TO BEHAVIOR	120
FIGURE 67: TEXTUAL NOTATION FOR TRANSITION	121
FIGURE 68: MIXED-TEXTUAL NOTATION FOR TRANSITION	121
FIGURE 69: GRAPHICAL ELEMENTS FOR MIXED-TEXTUAL NOTATION	122
FIGURE 70: TRANSITION-CENTRIC NOTATION OF A TRANSITION	122
FIGURE 71: ALTERNATIVE NOTATION FOR HISTORY STATE	133
FIGURE 72: PREDEFINED SET OF RANDOM CLASSES	157
FIGURE 73: UML CS PROFILE TIMER CONCEPT	159
FIGURE 74: NOTATIONS FOR STARTING TIMER	160
FIGURE 75: NOTATIONS FOR READING TIMER'S STATUS	161
FIGURE 76: NOTATION FOR RESETTING TIMER	162
FIGURE 77: DATA TYPE METAMODEL	165
FIGURE 78: PRIMITIVE DEFINITION WITH INFIX OPERATIONS	166
FIGURE 79: UNION TYPE AND IMPLICIT PRESENT FIELD	173
FIGURE 80: STRUCT TYPE WITH OPTIONAL FIELD	174
FIGURE 81: MAPPING SPECIFICATION BY OCL	178
FIGURE 82: OCL CONSTRAINTS ON ASI COMPOSITE OBJECT TREE	181
FIGURE 83: ABSTRACT GRAMMAR MAPPED TO METAMODEL	182
FIGURE 84: PROVISIONAL APPROACH MAPPING UML CS INPUT WITH FROM CLAUSE TO SDL	192
FIGURE 85: USING XSLT PRINCIPLE	203
FIGURE 86: RSVP AGENT TYPES	208
FIGURE 87: RSVP SIGNAL AND SIGNALLIST DEFINITION	209
FIGURE 88: RSVP DATA TYPES AND INTERFACES	210
FIGURE 89: UML CS SYSTEM OF RSVP MODEL	211
FIGURE 90: UML CS OVERVIEW OF NF BLOCK	212
FIGURE 91: STATE MACHINE OF NI PROCESS	213
FIGURE 92: ACTIVITY DIAGRAMS <i>INIT</i> AND <i>UPDATERT</i> OF NI PROCESS	214
FIGURE 93: STATE MACHINE OF NR	215
FIGURE 94: ACTIVITIES <i>INIT</i> AND <i>PATHCOMPLETED</i> OF NR	215
FIGURE 95: FIRST PART OF STATE MACHINE OF RSVP PROCESS	216
FIGURE 96: SECOND PART OF STATE MACHINE OF RSVP PROCESS	217
FIGURE 97: STATE MACHINE OF <i>FORWARDING</i> PROCESS	218
FIGURE 98: STATE MACHINE OF <i>ROUTING</i> PROCESS	219
FIGURE 99: <i>CALLFINDROUTE</i> AND <i>FINDROUTE</i> METHOD ACTIVITY	220

1 Introduction

Due to the huge complexity of modern software systems, it is required to specify precisely what a software component should do and how it should behave. If the final implementation deviates from the expected behavior, its use or its communication with other software components may fail. In a software development process, the *specification* describes the expected behavior of the software; the *description* describes the actual behavior of that software. The actual behavior is the implementation. A specification is a technical contract between developers and users or clients. It is mainly intended to provide them with a mutual understanding of the software and is used to guide the development and the use. This also applies for the development of communicating protocols as they are merely implemented in software.

Today, most protocol specifications are carried out in natural, informal language because it is easy to understand. However, experience has shown that specifications in natural languages can be vague, verbose and ambiguous. That means they can be interpreted in more than one way. A specification is formal if its meaning (its semantics) is unambiguous. Special languages, known as formal description techniques (FDTs), have been developed for the unambiguous specification and description of software. Hence, FDTs are distinguished from formal languages by having a formal syntax *and* a formal semantics. This stands in contrast to (semi-)formal languages such as Java or C++ which only have a formal syntax. FDTs are based on rigorous formal methods and offer the means for defining unambiguous specifications of network services and protocols in a more comprehensive and precise way than those done in natural language. In addition, they provide a basis for analysis, verification and validation of a specification before protocols are implemented. They can be used at the requirement stage to capture the user requirements. During the stage of analysis and design, FDTs can be used to describe abstract designs as well as detailed designs. There are several tools available for full or semi-automatic refinement from the formal specification into abstract and concrete designs and further into implementations. During the test stage, test cases can be generated from the specification in a systematic manner.

The Specification and Description Language (SDL) [ITU02a] is an example of a formal description technique. It is developed and maintained by the International Telecommunication Union, Telecommunication Standardization Sector (ITU-T). Due to the intuitive graphical elements, it has been widely adopted within the telecommunication industry. It aims at the unambiguous specification and description of the behavior of reactive and distributed systems and focusing on the object and state machine view of systems. SDL is mainly used in the design stage where an SDL specification defines the architecture and behavior of a system. SDL specifications can range from being abstract to concrete that can be simulated and validated automatically. Given a concrete specification, existing tools such as Tau [Tau] can generate implementations for different platforms.

Presently, the Unified Modeling Language 2 (UML) [OMG05a] is a collection of several semi-formal standard notations and concepts for modeling software systems at different stages and for several views on the same system. It is standardized by the Object Management Group. The UML semantics is described in natural English language. It includes semantic variation points that leave some semantics issues deliberately open. Therefore, UML specifications cannot be completely simulated, validated or executed, as the UML is too imprecise to fulfill this task. This lack of formality in UML is beneficial at early, conceptual stages of a development process. However, this desirable property turns into a drawback when the stage for simulation, validation and implementation is reached where SDL is well suited. In practice, UML can be made more formal by binding semantic variations in the UML language and providing a more precise behavior either in a tool or in a language profile.

The main goal of this thesis is to bridge the gap between the requirement and analysis phase and the design phase by combining the strengths of UML and SDL. While UML features multiple viewpoints on the same system, informal object models and property model views, SDL offers detailed formalized object models with respect to execution semantics. When re-using existing experience with SDL, UML and SDL can be combined so that one benefits from the advantages of both languages. In this thesis, an approach has been developed for generating full (behavioral and structural) SDL designs from UML-based system specifications by means of an extension to the UML, a *UML profile*. The application of this profile enables to specify and describe the structure and behavior of communication and signaling protocols by using the UML. It defines formal constraints to the modeling elements of the UML and binds the UML semantics to SDL by a formal mapping specification.

One of the design challenges of the profile is the selection of intuitive, efficient language concepts that are required for a specification and description of communicating systems. One might choose to devise language concepts from scratch. Presumably, this will lead to a cumbersome evolution process like other formal description techniques have already undergone. Another way is to examine and analyze an already well-established and accepted formal language for communication protocol engineering. This language could be used as the basis of the new profile. Elementary language constructs and concepts, which are already present, could be re-used within the UML so that a consistent profile architecture can be developed. Furthermore, high-level language elements could be added which provide a more abstracted view on modern communicating systems engineering. The latter approach has been adopted in this thesis.

SDL is considered a formal language and has been used for almost 30 years in telecommunication industry and practice. The long experience gained from SDL has influenced the development of the profile described in this thesis. SDL is used as the basis, streamlined and adapted for today's needs. It can be noticed that most SDL tool vendors have only partial implementations of the current SDL standard, named SDL-2000. One of the reasons for this could be the high complexity introduced by recent versions of SDL. Additionally, it is argued that many features provided by SDL are only required by a minority of users [She05]. Despite these concerns, SDL is the first language for specification, design and development of real time systems and in particular for telecommunication applications. However, it is not well suited for current and upcoming communication protocol engineering for packet switched networks. Such networks introduce typical effects like changes of the communication path, message losses and mobility of terminals. Furthermore, this includes protocol specifications for multi-hop overlay networks and multi-hop signaling tasks. Current packet switched networks like the Internet and mobile wireless access networks with devices constructing mobile ad-hoc networks [MM04] or overlay networks [Dan06] require new methodologies when modeling protocols. In this thesis, SDL is analyzed according to these criteria.

As a result of this analysis of SDL, the UML Profile for Communicating Systems (UML CS) and its additional concepts are developed. This UML profile enables the use of SDL concepts tailored for Internet communication and signaling protocols within UML notation by providing a formal mapping. This provides a means for UML-based specifications that can be analyzed, simulated and validated before the necessity of an implementation.

1.1 Scope of this Thesis

The scope of this thesis is to define a UML 2 profile that enables the unambiguous specification and description of Internet communication and signaling protocols. This comprises the following contributions:

1. An analysis of the Specification and Description Language (SDL) for its suitability to specify and describe Internet communication and signaling protocols.
2. A semi-formal definition of the UML Profile for Communicating Systems (UML CS). All constraints are defined by a formal specification language.
3. A definition of the profile's semantics by providing a formal mapping specification to SDL-2000.
4. A proof-of-concept implementation which maps UML CS-based models to a full structural and behavioral SDL design specification.

The profile described in this thesis has several features and concepts taken from SDL-2000. Some SDL language parts are omitted because they are rarely used or very complicated to use (this is described in Section 5.2 in detail).

A mapping to SDL is implemented by means of standard UML diagram interchange, the XML Metadata Interchange (XMI) [OMG05e]. Although this profile is based on SDL, it is not limited to SDL only. The profile's concept is based on extensibility. It offers some extension points that are incorporated to allow mapping to other formal description techniques (FDT). Moreover, the profile offers a number of additional high-level language elements which are defined especially for Internet communication and signaling protocol engineering. To summarize, this profile features a combination of SDL notation and semantics together with a flexible architecture to support multiple formal description techniques and important high-level features for communication protocol engineering for the Internet. Furthermore, this profile is designed for its applicability to several UML modeling tools that support the UML 2 standard with level three compliance¹ and the XMI 2.1 standard for UML diagram storage [OMG05e]. This broad applicability is a further challenge, as the UML standard specifies the abstract syntax and only provides recommendations for a concrete notation. As most tools differ in the implementation of the concrete syntax, this profile proposes several alternative structural, graphical and textual notations. This allows specifying and describing Internet communication and signaling protocols by a number of UML 2 modeling tools.

1.2 Related Work

The combined use of UML and SDL is not new. A number of different proposals exist to use UML and SDL together where UML shows its weaknesses and SDL its strengths [Bjo00].

The current ITU-T Z.109 [ITU99] recommendation in force at the time of writing this thesis (mid-2006) imports SDL into the UML 1.3. It makes use of the extension mechanisms available in the UML by defining a mapping from several UML elements to the corresponding language elements in SDL. To be more precise, it defines a one-to-one mapping between a subset of SDL and a specialized subset of the UML [Mol00]. This approach is very useful for system designers who are already familiar with both languages. However, for those unfamiliar with SDL, they might need some practice in order to use the Z.109 specification. Thus, the steep learning curve for this method makes it difficult for novices to use efficiently. Nevertheless, the strength of both worlds can be combined, the formal part of SDL and the number of views of UML.

¹ UML compliance levels are specified in [OMG05a]

It also has to be noted that the Z.109 recommendation is not part of the Object Management Group (OMG) specification of UML, but it is an independent, specialized version of UML to enable the use of SDL and UML together. Currently, the modeling tool Tau is available on the market that supports translation of UML into SDL based on the Z.109 recommendation [Tau]. At the time of writing, a *UML Profile for SDL* [ITU06a] is being developed by the ITU-T as a revised version of Z.109 which is based on the UML 2 standard [OMG05a]. In addition, the European Telecommunications Standards Institute (ETSI) had a work item for a *UML Profile for Communicating Systems* [ETS05] until 2005. It became a significant contribution to the ITU-T work and is now a joint work between both organizations. The ITU-T profile describes constraints and semantics of SDL by informal mapping rules. Besides many other conceptual issues, this is a major difference between the ITU-T work and the profile described in this thesis.

In [VE99], there are several scenarios presented for combining SDL-96 and UML 1.3 in which a mapping between the two notations is defined including behavior models. Very similar, the use of UML 1.3 and SDL-96 is described by a bidirectional mapping in [Ver01a, Ver01b]. It describes a set of rules that allow mapping from UML models to SDL models and vice versa. In [GG03], an approach is presented for the syntactic and semantic alignment of SDL and UML for the upcoming harmonization of both languages. However, this approach does not show a formal, complete mapping at all. In [BJ98], a methodology has been introduced for describing real-time systems using the UML 1.3. In this approach, concepts from *Real-time Object-oriented Modeling (ROOM)* [SGW94] are mapped into the UML by providing an extension to align both languages. It does not focus on the definition of a mapping from the UML into SDL, but it can easily be extended for a mapping from the *ROOM/UML* elements into SDL. Another mapping from SDL into UML 1.3 has been presented in [BJ00] based on informal mapping rules. In [Bra98], the UML is used for the modeling of real-time systems. The concept is also derived from *ROOM*. In [BK03], an overview of architectural design with the UML 2 is given illustrating a general approach of UML-based system description. However, the description of behavior or detailed execution semantics is not part of this work. A UML profile for modeling and formal validation of real-time systems is presented in [ACL+04]. It defines a profile extending the UML 1.5 and enabling a mapping to LOTOS [Hog89], a formal description technique, with real-time extensions [DBS95]. It is being updated to the UML 2. The *Integrated Method (TIME)* [Bra99] combines the UML for architectural modeling together with Message Sequence Charts (MSC) to depict the interactions between objects. It defines informal mapping rules for the mapping from UML to SDL based on the Z.109 recommendation which results in a complete system specification in terms of architecture and behavior. However, the mapping depends on manual interaction and therefore, is prone to errors and inconsistencies. Another mapping from UML combined with Message Sequence Charts (MSC) to SDL is presented in [BKV02]. It defines a mapping from standard UML elements to generate the SDL architecture and by adding the MSCs, it results in a complete specification of a system. A further approach is implemented in the *IF Framework* [BGO+04]. This framework is a toolset that maps UML and SDL specifications into an intermediate language and offers several tools for processing. However, a UML to SDL mapping is not defined. In [Wet04], a concrete implementation is presented to perform simulations based on UML 2 models that are mapped to SDL. For this, it uses the mapping functionality of the Tau tool. In [BOW00], a subset of the SDL action language is informally mapped to the UML action semantics.

In summary, all above-mentioned approaches are based on previous versions of UML and do not use the full potential introduced with the UML 2 except for the Z.109 revision, which is still being developed, and the modeling tool Tau G2. Tau is a UML modeling tool that binds UML 2 semantic variation points so that UML 2 can be mapped to an execution model based on SDL [Dol03a]. However, this mapping is proprietary and based on a restricted, early metamodel of the UML 2. Furthermore, none of these approaches focuses on communication and signaling protocol engineering for the Internet.

1.3 Novelty of this Profile

The UML uses multiple modeling paradigms and diagram types to model different aspects of a software system by providing multiple viewpoints on the same model. This enables to apply this language in almost all stages of a software development process. The system under development can be described in terms of abstract, platform independent to very concrete, platform specific models. The emphasis on this broad concept is reflected by the UML standard documents which define only very loose semantics described in natural language. However, the semantics deliberately provides multiple alternative or open semantics which may result in varying interpretations or ambiguous understanding. The concrete representation of the UML provides several alternative notations as well.

This *UML Profile for Communicating Systems* serves two general purposes: First, it provides a notation and tool support to a modeling language. Second, it provides more precise semantics for UML by applying specific semantics of a concrete language as a specialization to the UML. It can be stated that it is the first profile for communication protocol engineering built on the most recent UML 2 standard version and considers the revised items of the upcoming version 2.1 of the UML. It can further be stated that the profile described in this thesis contrasts to the ITU-T and ETSI profiles by using a formal language to define the constraints of the stereotypes and the profile's semantics. In addition, it provides a proof-of-concept implementation that allows to generate full behavioral and structural SDL design specifications from UML 2 models that apply this profile. This implementation utilizes the eXtensible Stylesheet Language Transformations (XSLT) [BPS+06] to process a UML model.

Furthermore, this profile does not only constitute a one-to-one mapping between SDL and UML, but it proposes several high-level concepts for the specification and description of communication and signaling protocols especially for the Internet. It is customizable, so that mappings from the same specification to other formal description techniques can be defined. It is aligned to work with several UML modeling tools that support the UML 2 compliance level three and have XMI version 2.1 support.

1.4 Structure of this Thesis

This thesis consists of ten chapters which are structured as follows: After this introduction, the following three chapters provide fundamentals for this thesis. First, the fundamentals of communication protocols and their engineering methods are briefly described in Chapter 2. It provides an overview of the basic terms and outlines the principles of communication protocols. Chapter 3 introduces the fundamentals of formal description techniques along with the Specification and Description Language (SDL) on which the profile described in this thesis is based. Chapter 4 briefly introduces the Unified Modeling Language (UML) and the extension mechanism that is used by this thesis, namely the UML profile mechanism. Furthermore, additional UML-related languages are introduced: the Object Constraint Language (OCL) for the unambiguous specification of constraints and the XML Metadata Interchange (XMI), a language used for diagram interchange of UML models.

Chapter 5 provides an analysis of SDL with respect to its suitability for specifying and describing Internet communication and signaling protocols. This analysis is driven by means of a concrete Internet signaling protocol implementation example, namely the Resource Reservation Protocol (RSVP). This chapter concludes with a discussion of the language constructs that are found to be missing or inadequate in SDL for the described purpose and are added to the profile described in this thesis.

Chapter 6 provides an informal overview of the UML Profile for Communicating Systems (UML CS) profile and a brief tutorial on its use. A semi-formal definition of the UML CS profile is presented in Chapter 7. Besides modeling elements with attributes and constraints, Chapter 7 gives an informal semantics and proposes graphical and textual notations for a complete structural and behavioral description including data types and timers. Chapter 8 provides the profile's semantics by specifying a set of formal mapping rules of modeling elements to SDL-2000 using the Object Constraint Language (OCL). In addition, the mapping from UML models to SDL is implemented by means of an eXtensible Stylesheet Language Transformations (XSLT) to demonstrate the feasibility and soundness of the profile's concept. Chapter 9 provides a case study of an Internet signaling protocol specification by means of a UML model that applies this profile. This model utilizes new modeling elements which are presented and discussed in Chapter 5.

Finally, a summary and an outlook of this thesis are given as a conclusion in Chapter 10. This thesis is completed by a list of used acronyms and a list of references.

2 Fundamentals of Communication Protocols

The requirements for a computer network can become very comprehensive. A computer network has to facilitate the common, reasonable, fair, robust and effective connectivity to a possibly high number of connected computers. In addition, the environment of a computer network can be of a highly dynamic nature. That means that components can be removed and attached at any time and that they are naturally heterogeneous. Communication protocols, which devices require to facilitate their communication, deployed in such a network have to deal with all these issues. New requirements introduced by new application technologies are also affecting the design of a computer network. This chapter introduces the fundamentals of communication protocols and their engineering. A more detailed introduction to this topic can be found in [Tan02, Koe03, PD03].

2.1 Network Layers

To cope with the high requirements and complexity of computer networks, a general engineering methodology has been developed which can be referred to as a generic network architecture. This can be conceived as a general guideline for the development and implementation of computer networks.

When a system becomes too complex to handle, system developers can introduce abstraction as a means of simplification. Such abstraction is used to define a unified model that covers the most important aspects of the system, but hides details that are not relevant from the current point of view. This model can be encapsulated into an insulated object that only provides interfaces to its environment through which it can be accessed. This conceals the concrete details how the object is internally implemented. Abstraction of details often leads to a layered design, especially in computer networks. The basic process model builds from the bottom layer up to higher layers, thus resulting in a stack of distinct network layers. The fundamental, low-level services provided by the used hardware components are subsequently abstracted by the introduction of layers offering services on a less detailed level. Services running within one of the higher layers use or implement services of lower layers to accomplish their tasks. So, as mentioned, such a layer can be conceived as an encapsulated object accessible and communicating only through well-defined interfaces. However, one of the major disciplines for the determination of a specific level of abstraction is to identify the most useful and suitable threshold: on the one hand, the chosen level of abstraction should abstract away from unnecessary details. On the other hand, the resulting design should be possible in a way that reusability is unimpeded and the implementation remains efficient.

The decomposition of computer network architecture into layer objects has two pleasant characteristics: firstly, the computer network can be decomposed into easily manageable components. It is not required to provide a huge, monolithic software architecture that has to render the combined functionality of several layers. But only to provide a set of layers with each of them simply focusing on the solution of a partial aspect of the overall task. Secondly, this results in a modular design. The addition of new services can easily be done by exchanging the appropriate layer implementation and re-using of the services provided by the remaining layers.

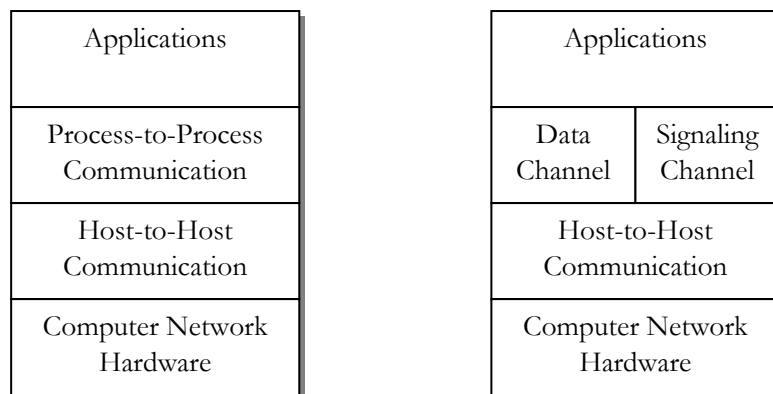


Figure 1: Computer Network Architecture decomposed into Layers

Figure 1 shows a possible decomposition of computer network architecture into layers. The layer stack on the left specifies two abstraction hierarchies by means of network layers defined between the applications and the underlying computer network hardware. The network layer placed directly on the Computer Network Hardware layer provides host-to-host communication to the upper layers. This layer abstracts (hides away) from the fact that two different hosts may be connected by a very complex network topology. The next higher layer provides process-to-process based communication and abstracts from the fact that the network may drop an occasional faulty message, for instance.

However, in many cases these layers are not placed in a simple linear order one over another. It is also possible that multiple kinds of abstractions are defined on a specific system layer which provide a different set of services to the higher layers, but also require to access the same abstractions on the lower layers. This is shown in the network layer stack on the right in the previous Figure 1. In this layer stack, the Data Channel provides services for transferring data. The Signaling Channel provides services for establishing a connection-oriented association to the communicating peer that, for instance, includes services like connection initiation and connection termination. A complete layer providing services to another layer is named a *service provider*. Network layers using services of a service provider are *service users*.

2.2 Communication Protocols

Network layers are commonly addressed by a numbering scheme. If there is a layer with a label value of (n) , there can be a layer with $(n+1)$, which is a higher layer (of abstraction), and there can also exist a layer $(n-1)$. The (n) -layer can communicate directly with the $(n+1)$ layer and the $(n-1)$ layer. In general, there is no further layer present below layer one. This specific layer may represent the lowest level of abstraction or, with other words, the real world (e.g. the hardware) [Sch03b].

The provision of a service at the service provider is done by means of the entities. Entities are active objects executing a behavior. They are contained within the service provider. They can interact with their environment by exchanging messages. The entity exchanges messages through several points of interactions, called *service access points* (SAP). Each SAP is uniquely mapped to one single entity; however, an entity can provide multiple SAPs for message exchange. An entity is able to accept tasks which are sent through a SAP. These tasks are specified by means of so-called *service primitives*. The service primitive is analyzed and the entity communicates with its peer entity by means of such a service primitive. A peer entity is an entity assigned to the same layer but on a different machine. In

order that two layers can exchange information there must be rules applied on a SAP. On a generic SAP, the entity on the $(n+1)$ layer transfers an *interface data unit* (IDU) to the entity on the layer (n) . The IDU itself is composed of a *service data unit* (SDU) and *interface control information* (ICI). The SDU is the part of information that is conveyed through the computer network to the peering entity of layer $(n+1)$. The control information is required to process the SDU, but is not part of the actual transmission. For the (n) -layer entity, the information which has been received by the $(n+1)$ -layer entity is encapsulated into a *protocol data unit* (PDU) and extended by a header that is, for instance, required in case of necessary fragmentation of the SDU. The PDU is used by peering entities to process the communication. This relation is shown in the following Figure 2:

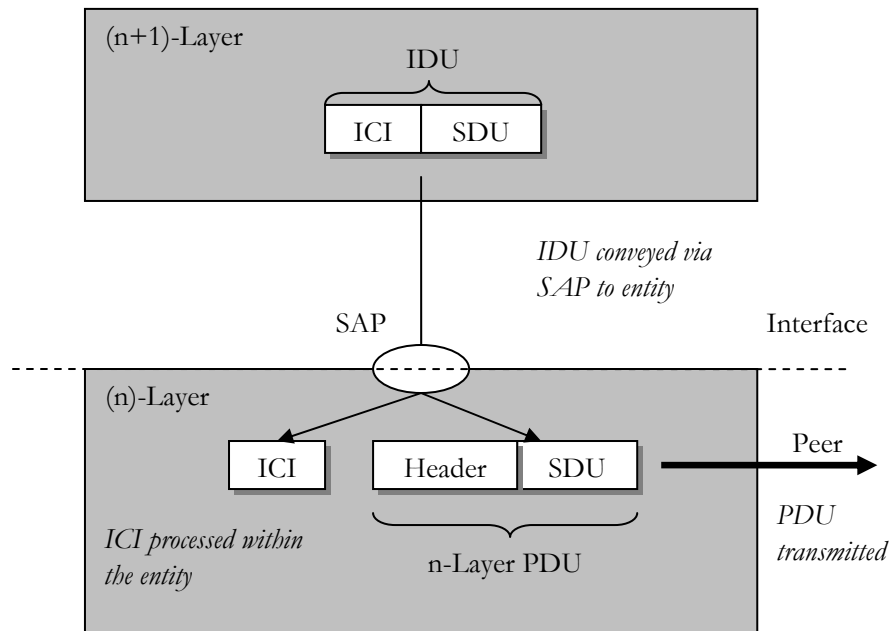


Figure 2: Communication between Two Layers

The communication between peering entities is processed by distinct rules. These rules are specified by means of a *communication protocol*. The following Figure 3 depicts the relation of entities, services, communication protocols and service access points: The service user accesses the service provider's entities via certain service access points. The service is provided between both peers. The entities communicate with each other by means of the communication protocol.

A communication protocol describes the interacting behavior of entities by specifying the timely sequence of messages that are exchanged. Furthermore, the format (syntax) and the meaning (semantics) of the messages are defined which are used by the communication protocol.

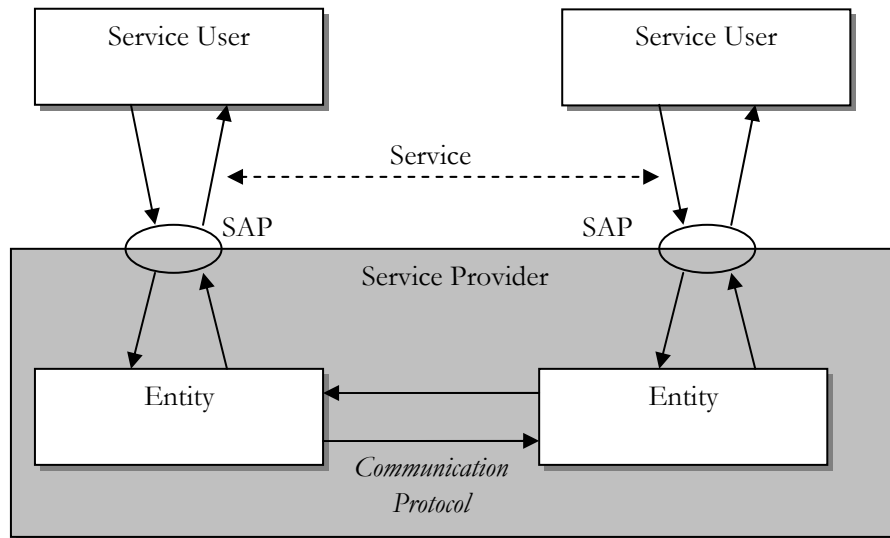


Figure 3: Interaction of Service Users and Entities

Communication protocols can be categorized in *symmetric* and *asymmetric* communication protocols. Entities using a symmetric communication protocol define a replicated behavior of communication. This would be the case if the entities allow bidirectional types of communication, for instance. Consequently, entities that use an asymmetric communication protocol have a different communication behavior based on their concrete role during communication. Examples for this are unidirectional communications where the transmission of data is only allowed in one direction.

2.3 Network Services

Computer networks are deployed to offer services to other users that allow exchange of programs, generic data, documents, files, remotely accessing other computers, requesting information from a computer, downloading an email or a web page from a web server. Newer applications also include service types such as video and audio conferences or instant messaging.

Such services can be distinguished into *symmetric* and *asymmetric* services. Similar to communication protocols, asymmetric services assign different roles to the communicating peers. The most common role designation is the *client-server* principle. A service is commonly hosted on a server and the client computer requests the service. The example could be a web server. A web client will request a certain, uniquely addressed web page from that web server that hosts the web page. This kind of service is mainly based on application services. In essence, the interaction procedure consists of the consecutive execution of a service request and service response.

Symmetric services provide a service on more than one service access point. In most cases, this relates to communication-based services for the exchange of data. Common network services rely on using communication based services to convey the service request and service response messages. For this purpose a so-called *connection* is established through which the communication takes places. This implies that the asymmetric services depend on the use of symmetric services.

Furthermore, services can be categorized in *connection-oriented* services and *connection-less* services. A connection-oriented service is comparable to a telephone call. This service initiates a connection, transfers the message and terminates the connection. In most cases, this includes the recovery from transfer errors or repeated delivery in case of failures.

On the contrary, connection-less services do not initiate a connection beforehand. The message which has to be transferred is simply sent out. There is no assurance, confirmation or reply that the message will reach its destination. If the message is lost or cannot reach its destination, it is silently discarded without any response. Additionally, the order of the messages which have been sent out may swap during transmission. Duplication of messages may also occur. This cannot occur using a connection-oriented service. This implies that services can also be classified according to their specific quality of service provision (Quality of Service, QoS).

2.4 Architectures of Communication Protocols

In computer networks and especially in the Internet, several network layers are used which implement a coordinated behavior with embedded communication protocols. This layered architecture defines the functionalities of a single network layer and defines the principles of communication between them. In general, this *communication architecture* is standardized by a panel of experts and by an associated standards institute.

Communication architecture specifications may be architecturally closed or open. *Closed communication architectures* are aligned for specific application areas. They are designed with focus on to the environment in which they are deployed. Most of them are vendor-specific or proprietary architectures. They are aligned to work with specific hardware or with specific network equipment only. One of the most prominent examples of closed protocol architecture is the *Systems Network Architecture* (SNA) by the International Business Machines Corporation (IBM). This architecture has provided the foundation for all IBM-based networks for many years. Due to the specific alignment of this architecture to the hardware, efficient adaptations and optimizations to the network environment were possible. However, heterogeneous networks were not possible to create. Nowadays, with the upcoming of the Internet closed architectures have almost completely been replaced by open architectures.

Open architectures of communication protocols support heterogeneous computer networks. Such architecture defines principles for the communication between different network nodes independently of their concrete deployment, operating system or network interface. All network nodes, which are conformant to the principles of the architecture, can be integrated in the overall computer network. These principles do not imply specific implementations. They only specify how systems have to behave through their external interfaces. One of the most well known open architectures is the Open Systems Interconnection (OSI) by the International Standards Organization (ISO) [ISO84] and the architecture of the Internet [Tan03].

Such architecture is described by means of *reference models*. Reference models act as a specification of communication architectures. They describe the amount of network layers, their individual functionality as well as the principles of interaction of the architecture. In the following, two examples of a reference model will briefly be presented: The OSI reference model and the TCP/IP reference model.

2.4.1 OSI Reference Model

The *Open Systems Interconnection Reference Model* (or OSI reference model for short) is the classic reference model for open architectures of computer networks. The model defines seven layers which have specific responsibilities. Whilst the four lower layers (at the bottom) are transport-oriented layers, the upper three layers are application-oriented layers, shown in the following Figure 4.

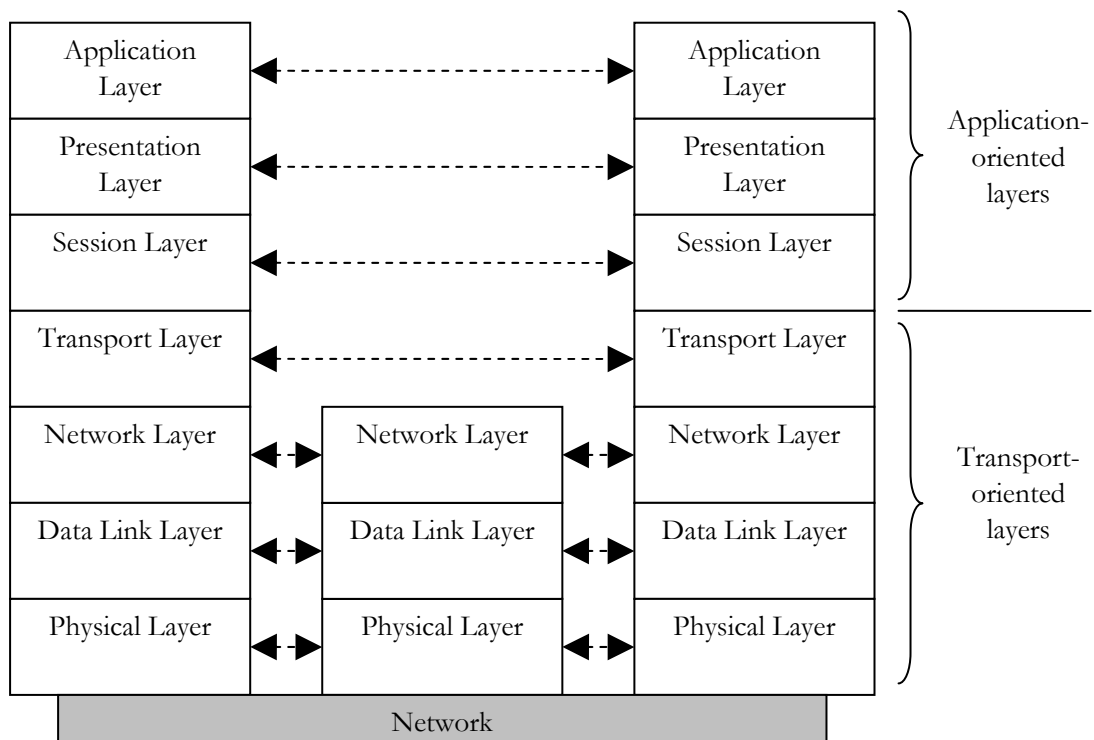


Figure 4: The OSI Reference Model

This figure pictorially outlines the architecture of three systems connected in a computer network. Not all of them have the entire OSI network layers implemented. The system in the middle only supports the lower three layers. Nevertheless, this is enough to participate in the communication up to a specific level. The OSI reference model describes the required functionality of the layers. The complete functionality is out of the scope of this thesis. Hence, the following is only considered as a brief overview:

The physical layer specifies the mechanical, procedural, functional and the electrical specification on the network medium. This includes specification for sockets, plugs and connectors including voltage values and the amount of data cable wires. The data link layer is responsible for segmentation, error detection and correct transmission of data frames to the next connected network system (point-to-point). The network layer implements the routing of data packets and therefore decides to which neighboring device a received data packet is to be forwarded (end-to-end). The transport layer establishes a logical connection between end systems (process-to-process). It abstracts from the actual network structure. The session layer is responsible for the correct establishment of each session, e.g. by re-synchronization and permissions to transmit. The presentation layer maps the information representation of data into a platform independent format so that heterogeneous systems can access the information. This may also include cryptographic or data compression functionality. The application layer finally provides services to the user by means of Service Access Points which are sometimes referred to as *Application Service Elements* (ASE).

OSI services which are provided by each of the layers are formally defined by a set of *service primitives* or *operations*. They allow a user or another entity to access and use a service. These primitives trigger the execution of a functionality of a service or configure the possible reaction on messages received from a peer entity. The OSI defines the following classes of service primitives: *request*, *indication*, *response* and *confirmation*. The *request* primitive triggers the execution of a specific functionality, for instance, the initiation of a connection or the sending of communication

data. The peering entity receives a notification of this event by the *indication* service primitive. The peer entity gives a *response*. This triggers the *confirmation* service primitive from the originating entity.

2.4.2 TCP/IP Reference Model

The *Transmission Control Protocol/Internet Protocol* (TCP/IP) architecture is the communication architecture of the Internet [Lie03]. It is not a reference model as described in the previous sections, but it is more commonly referenced as a *protocol architecture*². Protocol architectures define interfaces by means of the communication protocols, not by the communication architecture itself. Therefore, communication protocols in such an architecture are substitutable and can be implemented in other protocol architectures as well. Protocol architectures describe well-defined hierarchies of communication protocols aligned to specific application cases or environments. The TCP/IP architecture is sometimes referred to as a *protocol stack* being the concrete implementation of a protocol architecture [KR04].

The TCP/IP protocol stack defines the core *Internet Protocol (IP)*, consisting of the TCP protocol for connection-oriented transmission, the *User Datagram Protocol (UDP)* protocol for connection-less transmissions and the IP protocol for end-to-end transmissions. TCP/IP defines four network layers that are depicted in Figure 5.

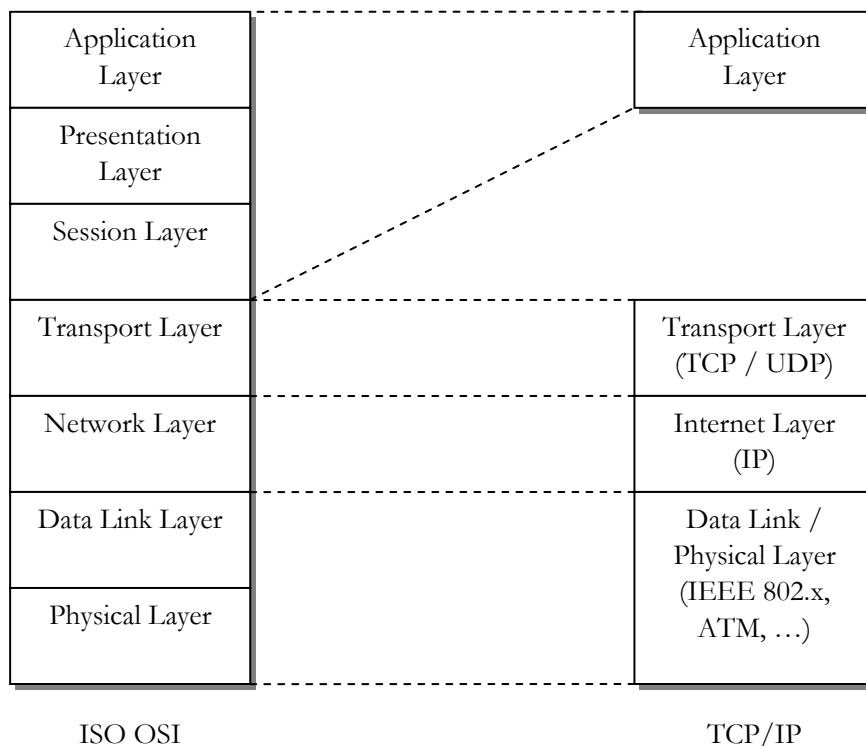


Figure 5: ISO OSI Reference Model and TCP/IP Architecture compared

² The term *TCP/IP reference model* can often be found in literature.

The most important layers are the transport and the Internet layer³ which are the only complete defined layers. The remaining two bottom layers depend on the application area and network environment. Their main task is to enable reliable transmission of data between two endpoints even in case of single intermediate route or connection failures. Contrary to OSI, heterogeneous networks were considered as a part of the architecture from the beginning. Therefore, IP was introduced for connection-less transmissions. While TCP enables reliable transmissions between two endpoints, the UDP provides unreliable transmissions between two endpoints which is more suitable in real-time applications like video or audio streaming. Besides other things, a difference between the transport and the Internet layer lies in the fact that TCP and UDP support process-to-process data transmissions while IP only supports host-to-host transmissions.

The neighboring upper layer of the transport layer is the application layer, skipping the session and presentation layer which are unknown to TCP/IP. For most applications, these layers have shown not to be essential or useful. The layers beneath the IP layer are not specified as well. This is deliberate as the underlying network architecture and technology may vary from network to network. In most cases, this requires a development of specific communication protocols which enable the interaction of TCP/IP with the specific network. This is also one reason why the TCP/IP is not a reference model in the same tenor.

2.5 Summary

In this chapter, the fundamentals of communication protocols and services have been introduced. In the first section, the concepts and benefits of the layered design for communication protocol engineering have been described. The communication principle and communication primitives between various network layers and entities have been outlined in the second section. The fundamentals of network services and the different service categories have been presented in the third section. The fourth section has described the principles and reference models of communication protocol architectures. This included the introduction to the well-known OSI network reference model and to the TCP/IP protocol stack.

³ The Internet Layer is also called Network Layer.

3 Description Techniques for Communication Protocols

For the specification and description of communication protocols, several different methods are available. As a communication protocol is primarily implemented by software, the usual software engineering methods can be used. However, due to the nature of communication protocols some characteristics can be well covered by means of formal description techniques. Formal description techniques utilize formal methods to describe a communicating system unambiguously. Formal description techniques differ from a conventional software engineering process in the way that a rigorous mathematical foundation and formalism is used for the specification and description. Formal methods provide several benefits compared to conventional programming techniques by providing the basis for proofing, verifying and validating a system description with respect to specific properties [Dol03b]. Further, it can be used for the (semi-)automatic refinement of a communicating system description into its final implementation. When the stage for testing is reached, test cases can be derived from a formal specification in a systematic manner. The following chapter introduces formal methods and concepts for the specification and descriptions of network services and communication protocols. It further introduces two formal description techniques, the Specification and Description Language (SDL) and Message Sequence Charts (MSC).

3.1 Specification of Services and Protocols

At first, the specification of a communication protocol provides the basis for the development process. The specification describes the requirements and the desired behavior of the communication protocol and its services. This specification acts as a reference for the actual implementation process, its accompanying validation process (e.g. testing) and the use of the service. The following Figure 6 pictorially represents the protocol engineering process.

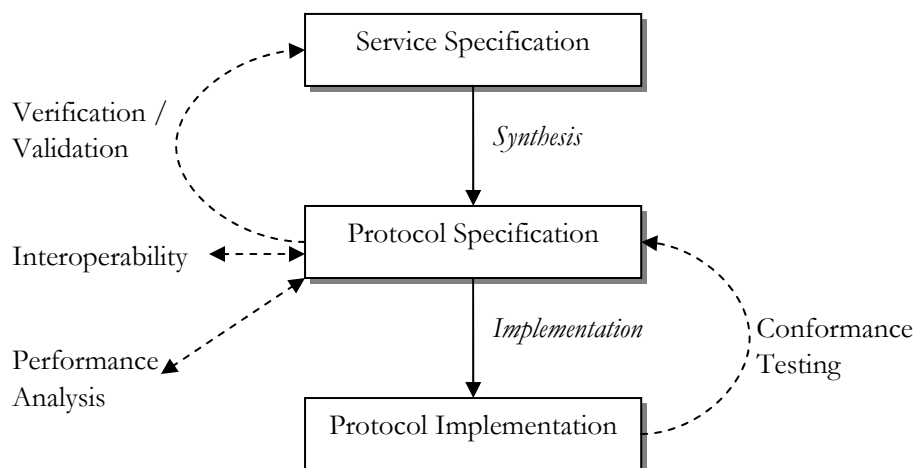


Figure 6: Protocol Engineering Process

The specification for communication protocols is merely identical to a common software specification. That is, the specification shall be *exact*, *unambiguous*, *complete* and *abstract* [Got93]. Exactness, unambiguosness and completeness are essential parts of a specification that acts as a reference for the

actual implementation. Without these properties being fulfilled the specification can only be given vaguely. Hence, a comparison of the prototype's behavior compared to its specification would not be possible in all cases.

Abstractness describes independence of the platform and implementation environment. Internal architectures and control flows can be neglected while the externally observable behavior is a vital issue. Therefore, the specification of communication protocols and services describes only functional behavior. Implementation-oriented aspects like the concrete addressing of multi-cast connections are typically not considered. This can be addressed in a subsequent implementation description. A specification facilitates several different implementations which are conformant to the specified functional behavior. These implementations are considered *conformant* with respect to their specification.

The main difference to common software specification is that the specification of communication protocols is twofold, as shown in Figure 6: First, a specification that describes *what* has to be provided (to a user) and second, a specification describing the method and the *way* this can be achieved. The first specification is the specification of the service. It is comparable to a common software specification. The second specification is the specification of the communication protocol. The latter represents the abstract implementation of the previous service specification. To be complete, each specification should include explicitly [Hol91]:

1. The *service* to be provided by the protocol
2. The *assumptions* about the environment in which the protocol is executed
3. The *vocabulary* of messages used to implement the protocol
4. The *encoding* (format) of each message in the vocabulary
5. The *procedure* rules guarding the consistency of message exchanges

It is interesting that the fifth point is the most difficult to design and the hardest to verify.

The service specification describes how a service user can access the provided services at the service access points. This includes the description of the services itself as well as the service primitives and parameters and the relations and dependencies between the parameters and the service primitives. This part of the specification is relevant for the service user. It defines how one can access the provided service. The protocol specification describes how the service has to be provided by the service provider. It describes the internal communication between the providing entities and specifies the message format of the protocol data units. Typically, this includes the specification of the communication between the (n)-entity and the ($n-1$)-services. This specification is relevant for the communication protocol developer who has to conduct the tests, perform inspections and implementations. It is of minor relevance to the service user.

Contrary to ordinary software specifications, a protocol specification describes the communication behavior between concurrent processes. This imposes effects like timing problems, race conditions and possible deadlocks [TS03]. The precise sequence of events cannot always be predicted. So, the amount of possible sequence of events can be so overwhelming that an exhaustive protocol analysis by manual code inspections or walkthroughs is virtually foredoomed. Hence, a more structured description of protocol's behavior is necessary. The description of communication protocols can be categorized into *behavior-oriented* and *communication-oriented* viewpoints [Koe03].

Behavior-oriented descriptions describe the protocol by means of the behavior of each of the involved communicating entities, e.g. by using the Specification and Description Language (SDL) which is described in detail in Section 3.4. The actual protocol communication is not described but can be derived from the behavior of the entities. This is the preferred viewpoint of formal description

techniques (FDT) as this is founded on several semantic models like finite state machines and process algebra. The actual message flow based on the protocol execution can only be figured out by a thorough analysis of this description. This is due to the nature of communication protocols as the focus lies more on the protocol's message exchange than on the behavior of the communicating entities.

The communication-oriented viewpoint describes the concrete communication exchange between all entities, e.g. by using Message Sequence Charts (MSC) which are described in more detail in Section 3.5. Contrary to the behavior-oriented viewpoint, where each entity is described independently, this description includes all entities. This results in an improved intelligibility of a communication protocol's design and concept. However, the semantic model is more complicated and the interactions of several protocol runs are hard to describe. Hence, communication-oriented descriptions are used in addition to behavior-oriented descriptions nowadays.

This thesis focuses on systems that are assumed to maintain an ongoing interaction with their environment. This environment could be the controlled system or other components of a communication network. Therefore, they are called *reactive systems* [Mer01]. Traditional models that describe computer programs as result of a computational function from given input parameters are virtually inadequate for the description of reactive systems. Instead, a much more concise description of the behavior of reactive systems is possible by transition systems or by finite state machine models. Reactive systems can be broadly classified into *distributed* or *concurrent* systems. Distributed systems have spatially separated subcomponents whereas concurrent systems share resources such as processors and memories. Distributed systems communicate by passing messages; concurrent systems can use shared variables [Ham05]. Analogously, a protocol is commonly specified as a set of processes. These processes interact with each other by exchanging messages over some communication channels. On a very low level of abstraction, a communicating process is usually described by means of a finite state machine. This formalism easily allows the expression of design criteria by defining desirable or undesirable protocol states and transitions. In some aspects, the protocol state can be conceived to symbolize the assumptions that each process in the system makes about the others. A finite state machine unifies the actions a process can execute, the events it expects to happen and in which way it will respond to such an event. There are a number of different formal methods available for the specification of communication protocols [AAL+99]:

- State Transition Models: Finite State Machines (FSM) [HMU92], Communicating FSM (CFSM), Petri nets,
- Programming Languages Models: Abstract Programs, Calculus of Communicating Systems (CCS) [Mil80], Communicating Sequential Processes (CSP) [Hoa04], Temporal logic (Labeled Transition Systems) and
- Hybrid Models: Extended FSM (EFSM).

Formal description techniques use these formal methods as a mathematical basis. For instance, the Specification and Description Language (SDL) incorporates an extended finite state machine approach [ITU02a]. ESTELLE uses extended finite state machines and an extended Pascal programming language dialect [ISO97]. LOTOS uses the calculus of communicating systems with abstract data types [ISO89, Hog89].

3.2 Model-based Development

The construction of software and protocols by means of modeling techniques is gaining a lot of attention, particularly with the recent release of the *Unified Modeling Language* (UML) in its second version. Furthermore, this becomes underlined through various initiatives in *Model-Driven*

Architecture (MDA) [OMG03b] and *Model-Driven Development* (MDD) [FGD+06]. The UML is designed as a top-down modeling method. It permits large complex processes and software products to be described in a standard, structured way. To a certain extent it can be explored by simulation. However, the primary goal that this could lead to the automatic production of software, in particular, the automatic generation of program code, has not consequently been achieved with the UML.

One of the most difficult challenges of software for embedded systems, for telecommunications and for reactive systems in general is to ensure that the behavior of the software is accurately controlled in all possible conditions. In particular, this includes the presence of unusual or unanticipated combinations of external stimuli and effects of design flaws. This area can be well expressed in terms of *finite state machines* (FSM). FSMs are built on a rigorous mathematical foundation and are much easier to design, to understand and to discuss than the corresponding program code. A number of modeling tools support the generation of program code from such FSM models expressed as state machines. Nonetheless, there are major difficulties: unless all the intimate details are defined in the model, the code cannot be complete. In the typical case, nothing more than some corresponding code skeletons or header files can be produced. It requires to complete the software by adding legacy or manually developed code. This is inherent in the top-down approach to design. Despite of these flaws, it is argued that model-driven engineering

- accelerates the development process (automation by formalization),
- increases the quality of software (by exhaustive simulation, validation and systematic test case generation),
- enables better handling of complex software (by abstraction) and
- improves conception and consistency of the overall program design.

On the other side, these potential benefits rely on good abstraction capabilities of the designers, require high efforts for rather small projects and later code changes may be cumbersome. However, models are a useful means to abstract from specific details of the system.

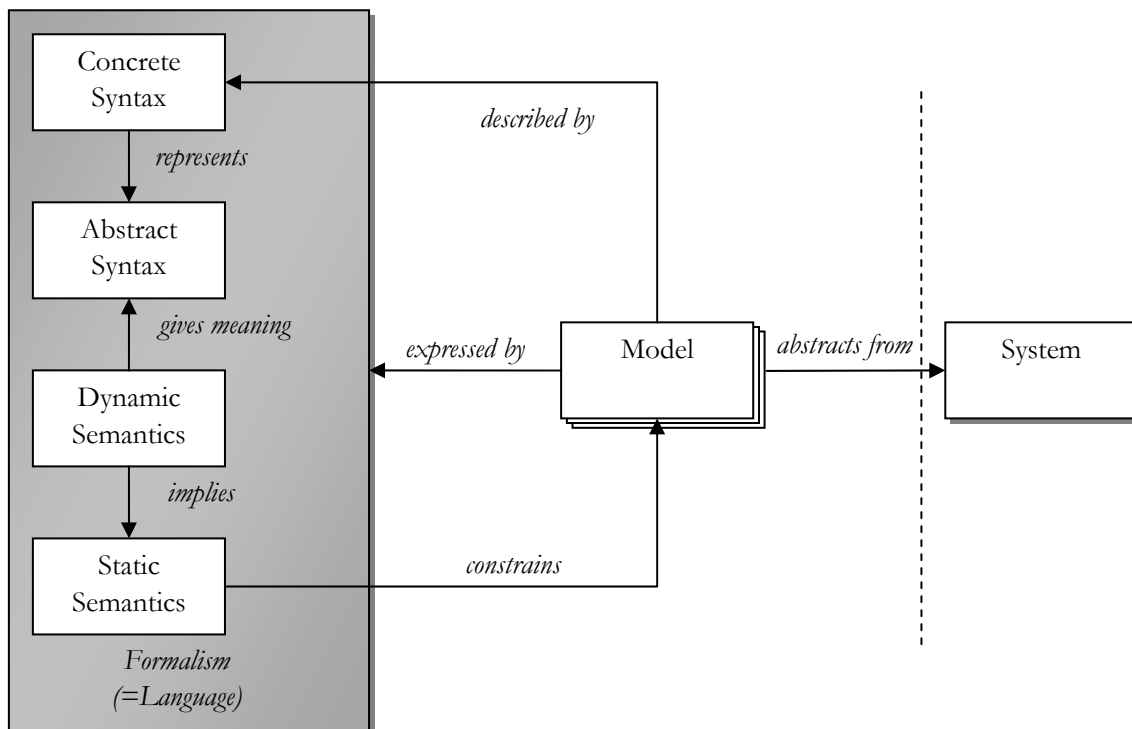


Figure 7: Concept of a Model

Models are expressed by a specific language based on a formal semantics and syntax. This language can be of a graphical or a textual representation. Figure 7 outlines the meaning of a model. There may be several interrelated models defined, but with different levels of abstraction, e.g. platform independent (PIM) and platform specific models (PSM). A model is expressed by means of a specific language. This language defines a concrete syntax acting as the intelligible, human-understandable notation. This concrete syntax is an instantiation of the abstract syntax. The abstract syntax abstracts away from specific details of the concrete syntax, as the concrete syntax is not essential for the definition or specification of expressions such as blank spaces, comments and delimiters. It represents the deep structure in terms of an abstract syntax tree. The dynamic semantics defines the run-time behavior of a model. As shown, this also implies additional constraints on the model which are summarized in the static semantics. Static semantics constrain the model statements at compile-time. The static semantics is implied by the dynamic syntax. This encompasses visibility rules, matching function signatures, compatible data types and so on.

3.3 Formal Description Techniques

Today, network services and communication protocols are specified informally in many cases. Prominent examples are the Internet standards (so-called *Request for Comments*) issued by the Internet Engineering Task Force (IETF). They are standard documents, composed of plain text in ordinary English language with additions of some textual character-based graphics to show network configurations and message exchange. Such a specification is merely composed of scattered descriptions of partial protocol runs of the entities. In most cases, this describes the behavior of the entities after some event has occurred. Then, the protocol developer has to reassemble the complete behavior of an entity from the single description parts. Nevertheless, informal specifications have the advantage that they can be easily accessed and understood. The disadvantage is that informal specifications tend to become ambiguous, imprecise and incomplete. Special occurrences or conditions are simply not considered. Multiple features interact in a way that was not anticipated or expected beforehand. Hence, different protocol developers freely interpret the specification at whim and consequently deviate in their final implementation. This results in incorrect and incompatible protocol implementations. Furthermore, computer-aided validation and testing of the development process can hardly rely on informal specifications.

The necessity for a formal description of communication protocols is commonly undisputed and accepted. A formal description is a description by means of a description method with a strong formal semantics which ensures an unambiguous interpretation of the description. The following Figure 8, taken from [Koe03], depicts this process.

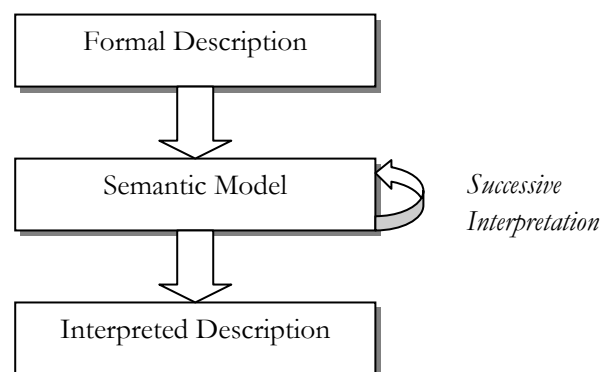


Figure 8: Interpretation of Formal Descriptions

The description of communication services and protocols is composed by the description of a communication behavior and the description of the protocol data formats for the service primitives and protocol data units. For data formats, several techniques for data descriptions have been developed. A brief overview is given in Section 7.7. The description techniques for communication protocols can be categorized in *constructive* and *descriptive* methods [Got92].

Constructive methods describe a communication protocol behavior by means of an abstract model. This model is executed by a virtual machine and it shows the behavior of the communicating entities. This description can be considered as near-to-implementation as the protocol is described by means of a more abstract, higher-level protocol description. This results in direct support for designing, validating and implementing the communication protocol. Derived from the specification and description, executable prototypes can be generated automatically. However, specific requirements for a communication protocol like liveliness cannot be described explicitly and have to be verified by the chosen description model.

Descriptive methods do not describe the behavior of a communication protocol, but they describe properties that have to be fulfilled by the implementation of logical expressions. The main advantage is that properties of the protocol can be described without any implementation details. Therefore, one can validate them independently.

3.3.1 Formal Languages

As discussed in the previous sections, formal languages can play an important role in the software development process and in the development of communication protocols. The nature of formal languages is to have an exact definition of the expressions that belong to the language, the *syntax* of the language, and what these expressions have for an exact meaning, the *semantics* of the language. Furthermore, the *pragmatics* of a language is important. For programming languages, pragmatics includes issues such as ease of implementation and programming methodology [SK95].

A specification is formal if its meaning (semantics) is unambiguous. Formal Description Techniques (FDT) are languages that are distinguished from formal languages by having a formal syntax *and* also a formal semantics. This makes a difference to formal languages such as Java or C++ which only have a formal syntax. They are rather implementation languages than description languages. It is generally accepted that it is essential to produce a thorough and exhaustive system specification and design for the successful development of a system. Specification languages can help to accomplish such task if they are capable of satisfying the following needs [Abs93, Ver01b]:

- unambiguous, clear, precise and concise specifications,
- a thorough and accurate basis for analyzing specifications,
- a basis for determining whether an implementation is conformant to the specifications or not,
- a basis for determining the consistency of specifications and
- a translation for generating applications without the need for manual coding.

The following sections briefly give an outline how syntax and semantics can be defined by formal means. However, this covers only very basic parts. A more detailed description can be found in [HR00, EMS00, Pri01].

3.3.2 Formal Methods for Syntax Description

A formal grammar is a way to formally define a syntax. A grammar G is defined by the quadruple of a set of terminal symbols T , a set of non-terminal symbols N , a distinguished starting symbol s that is an element out of N and a set of grammar rules R (called production rules). A grammar rule consists of left-hand-side rules LHS and right-hand-side rules RHS . Both sides can contain a mixture of non-terminal as well as terminal symbols which are concatenated together. The grammar allows to recursively substitute a non-terminal symbol with each other with respect to the given rules. This process yields a set of possible expressions which define the syntax of the language specified by this particular grammar. For example, the following formal grammar defines an arithmetic binary expression consisting of multiple '0' and '1' with a '+' and '-' signs and open '(' and closed ')' brackets:

```
T={ '0', '1', '(', ')', '+', '-' },
N={ expression, literal }, s={ expression }
R={
<literal, '0'>, <literal, '1'>,
<literal, '0' literal>, <literal, '1' literal>,
<expression, literal>, <expression, literal '+' expression>,
<expression, literal '-' expression>;
<expression, '(' expression ') '>
}
```

An easier way for the notation of such a grammar is e.g. the Backus-Naur-Form(alism) (BNF). The BNF defines the following notational template for a single grammar rule:

```
LHS ::= RHS
```

The pipe symbol '|' given in the RHS specifies alternatives. In addition, there exists an extended variant of BNF, the Extended Backus-Naur Form (EBNF). This notation introduces some additional symbols for denoting recursions. For instance, zero to infinite recursions are denoted using the '*' symbol; the '+' symbol for one to infinite recursions. The square brackets '[' and ']' specify an optional rule expression. That is, the symbols inside the square brackets may be chosen to be left out or to be included into the production rule. Per convention, the start production rule in BNF expressions is the topmost one on the left. The grammar from above specified in EBNF will therefore look like:

```
expression ::= literal | literal ( '+' | '-' ) expression
              | '(' expression ')'
literal ::= ( '0' | '1' )+
```

3.3.3 Formal Methods for Semantics Description

Semantics of a language describes the meaning of a syntactically correct statement in the context in which it is expressed. Formal semantics can provide a complete, rigorous definition of a language. The most important benefit is that this definition can act as the compulsory reference for system designers who want to understand intimate details of the language and also for implementers of the language itself by providing a standard against which the compliance of an implementation may be assessed [HB00].

Providing a robust definition can help to minimize the possibility that different implementers might interpret the definition of the language differently. If a language definition reveals ambiguities, one may face the possible consequence that a user's program might run fine on one implementation, but

not on another language. A formal definition is typically much more concise than a corresponding description in English text. Thus, the main advantages of formal definitions – in contrast to their informal counterparts – are that they are inherently precise and unambiguous by using mathematical statements. Albeit, the downside is that the basics and benefits of formal semantics are not very widely known and understood yet. Nevertheless, a formal semantics can provide the mathematical foundation that enables to prove certain properties of programs. In order to formulate claims about programs a vital prerequisite is have a precise mathematical definition of the language.

There are two different kinds of statements possible one might wish to prove: statements about the language itself and statements about particular programs. An example of the first kind may be the claim that null pointers and type errors cannot arise at run time. Examples of the second kind include proofs of correctness of programs. Normally, the term *correctness* means that a program conforms to or behaves according to a given (formal) specification. For this purpose, formal semantics provide logical and conceptual tools needed for proving and defining precise statements of correctness. However, being able to prove the correctness of programs by mathematical means has been discussed and researched for many years now, but it had still not become practical for sizeable programs.

The definition of semantics for specification languages has been researched over a long time. For mathematical languages well-known semantic definitions have been developed, but the definition of a semantics of a specification language is considerable more difficult. Specification languages have many special cases and a dynamic semantics which gives a new meaning to the statement based on the state the current interpretation has reached. There are several methods available for specifying semantics. Generally, they can be divided into the following categories:

- Informal semantics
- Translational semantics
- Operational semantics
- Denotational (or functional) semantics
- Axiomatic semantics

Informal semantics do not use formal methods for the semantics description, but in most cases (e.g. for C, Pascal, Ada, Java) these semantic definitions are written of carefully composed texts using natural language. Rapidly, this can become a very complex task because such standard documents have to describe not only every individual construct in the language, but also all possible combinations and interactions of these constructs have to be respected as well. Subtle issues can arise from several feature interactions and tend to augment rapidly. Hence, for very sizable and complex languages it is very difficult to take each possible construct and combination into account while keeping the description sufficiently precise and free from ambiguity. This makes it very hard to be certain that all possible interactions have been considered and thus, possibly resulting in a huge, long-winded and incomprehensible document. It is common that several additional examples are given which shall clarify the meaning and any possible ambiguities which may arise by reading the description. In spite of this effort, it is often prone to be interpreted in a wrong way as ambiguities may arise differently based on the respective reader.

Translational semantics is a special case of the definition of semantics. Translational semantics do not specify semantics itself, but translate a statement of a source language to a statement in another target language. The semantics for the target language statement can be mapped backwards to give a semantics to the source language statement. This is only possible and meaningful if the target language has a strong semantic basis. This may also introduce a steep learning curve as not only a single language and its semantics description has to be known. This will require to acquaint oneself both to

the source and target language and as well as to the translational semantics description and the semantics description of the target language. However, for target languages which are considered well known and understood this can be a suitable way to bind a semantics description to a language. This may allow that this language can also be understood quite fast.

Operational semantics are very close to a concrete implementation. The fundamental idea is to define the execution by a virtual or abstract machine which interprets each instruction. An example for such a machine is the *Turing Machine*. By means of the machine's description how it proceeds in execution, the behavior of the program – the semantics – is specified. The machine defines a transition function that specifies a subsequent state for each active state. Therefore, the semantics of the program is specified through the sequence of states traversed from the initial state to the final state.

Contrary to operational semantics, denotational semantics do not specify execution steps of an abstract machine. But it specifies a functional correspondence between the program variables which are modified through the execution of the program. This denotes how the values of the variables are influenced by the construction of the programming language. For this purpose, functions are defined based on the domain of the syntax and the co-domain of the semantics. State changes triggered by the program are described by these functions.

By axiomatic semantics, an axiomatic system is formalized to predicate state changes by means of logic equations and model theory. The entities of the language and their inter-relations are specified.

These three styles do not compete. They are mutually complementary and focus on different aspects of a language. Operational semantics are the most intuitive and useful standard for implementers. Axiomatic semantics are most suitable for program verification. Denotational semantics strike a balance between operational and axiomatic semantics by providing the logical and conceptual link between both. It encourages the well-structured design of both programming languages and program logics. For instance, the semantics definition of the Specification and Description Language 2000 (SDL-2000) – introduced in the following section – is defined by means of a translation [GGP03] from an SDL program to an Abstract State Machine (ASM) program [Gur88]. The latter three semantic description styles are formal as they are founded on a strong mathematical basis. By the use of formal means, this allows to derive further answers from these semantics such as if there are language constructs that must not be used together or if there are language constructs which can be composed by others. Additionally, any behavior of the specification can be covered and analyzed by mathematical means. Some properties that are commonly examined by validation and testing are:

- *Absence of Deadlock conditions*
The system never enters a state that cannot be left due to a missing or occupied resource.
- *Absence of Livelock conditions*
The system never enters cycles that cannot be left due to a missing or occupied resource.
- *Code Coverage*
Each statement defined in the system can potentially be executed.
- *Liveliness*
Each state of the system can be reached from the initial state.
- *Robustness*
The system can react to unexpected, unusual or missing events.
- *Termination*
The final state – or an idle state for cyclic systems – can always be reached.

- *Recovery from Failures*

The system can recover to a normal state within a limited time after an error has occurred.

It is well-known that these questions are non-decidable in general. That is, it is proof that a general algorithm to verify these properties cannot exist for all program-inputs it tries to analyze (The *Halting problem*, [HMU02]). However, these very useful properties come by means of a formal semantics. The downside of formal semantics is the fact that it is hard to understand for the inexperienced programmer and tool developer, because it requires a good knowledge of the underlying mathematical means.

3.4 The Specification and Description Language

3.4.1 Introduction

The Specification and Description Language (SDL) is an executable, object-oriented, formal modeling language intended for the specification and description of telecommunication systems and telecommunication protocols. The SDL standard is maintained and published by the International Telecommunication Union – Telecommunication Standardization Sector (ITU-T) which is known as the Recommendation Z.100 [ITU02a].

In the early 1970's, the first efforts began to develop a specification and description language within the Comité Consultatif International Téléphonique et Télégraphique, CCITT (later renamed to ITU-T). This effort was pushed by the upcoming era of digital, computer-driven devices in telecommunication networks that supplanted the old analog and traditional electronically engineered devices successively. Because of this introduced new technology, an immense amount of new services could now be offered to customers. Thus, a programming language was needed to cope with the increasing complexity which was accompanied by this new era. The first version of SDL was published as Recommendation Z.100 in the year 1976 (so-called *orange book*). This version consisted of a small collection of standardized graphical symbols for modeling event-driven, reactive systems based on finite state machines. However, the semantics of the symbols and how they could be used together was not explicitly explained. The ITU started their activities on a four-year basis on the maintenance and improvement of SDL. This effort resulted in the release of a new recommendation at the end of each period. Consequently, in 1980 an improved version of SDL was presented: SDL-80 (the *yellow book*) addressed the problems of the first version and added informal semantics to the notations. Structuring of the state machines was introduced by means of the concept of systems and blocks and procedures. SDL-84 (the *red book*) introduced data and data types for application in state machines and signals.

A noteworthy milestone was reached with the release of SDL-88 in the year 1988 (the *blue book*) by including a formal semantics. This formal semantics was based on a combination of the Vienna Development Method (VDM) [BJ78] metalanguage Meta-IV and a Communicating Sequential Processes (CSP) [Hoa04] based communication mechanism. Unfortunately, the CSP communication extensions were based on informal descriptions. This was an often-cited flaw of the formalism [Hin98, BGM+02, Gra03, Pri03]. This version of SDL already contained many features which are still available in current versions. This includes asynchronous communication, hierarchical structuring of the system, data types and an extended communicating finite state machine. Next remarkable evolutionary steps of SDL took place with SDL-92 and SDL-2000. SDL-92 introduced object-orientation to SDL and SDL-2000 completed the step to introduce object-orientation paradigms. The SDL-96 version was primarily a bug-fix and an update of few language concepts.

This latest version of SDL has been combined with object-oriented modeling techniques and improved on its use for the automatic generation of implementations by means of several new language features and concepts such as [Ree00, FHL+00]:

- exceptions and exception handling,
- new data model, including object data and direct support of the Abstract Syntax Notation number 1 (ASN.1 – a platform-independent, descriptive language for defining data structures) with SDL,
- a unified structuring concept for blocks, processes (agents),
- composite states and state aggregations and
- interfaces, class symbols as references, associations.

However, not only new features were added but also some SDL features have been undergone a major streamlining. Additionally, a new underlying semantics was introduced which removed the value semantics by a reference semantics based on Abstract State Machines (ASM) [Gur88]. An ASM formalizes the discrete dynamic behavior of a system in terms of executions of an abstract machine. The system's behavior is expressed in the form of a deterministic state transition system over abstract data structures by means of a simple, but universal language.

SDL has its main application area in the specification and description of telecommunication systems. In fact, that is according to Z.100 [ITU02a]:

- A *specification* of a system is the description of its required behavior,
- A *description* of a system is the description of its actual behavior – its implementation.

In SDL, as there is no distinction between its use for specification and its use for description, the term specification is used for both – required behavior and actual behavior.

3.4.2 Language Concept

For the definition of the syntax, semantics and the properties of SDL, different metalanguages according to their strengths have been used with respect to the particular needs [GGP03]. The SDL language definition consists of the following parts:

- syntax,
- an informal semantics written in English language and
- a formal semantics based on Abstract State Machines (given in annexes to the standard)

The syntax of the language is for itself given in three variants, an *abstract grammar*, a *concrete textual grammar* and a *concrete graphical grammar*. The abstract grammar abstracts away from such helping constructs like delimiters, separators and keywords and only summarizes the vital attributes of a language object. For SDL, two different abstract grammars, AS0 and AS1, are defined. The AS0 is closely related to the concrete grammar. It is obtained from the concrete syntax by omitting the noted details. The AS1 is obtained from the AS0 applying transformations followed by a mapping [Pri01].

The abstract grammar AS1 is given in a form of a named composite object defining a set of sub-components and with the previously introduced Extended Backus-Naur Form. For example, the abstract syntax of an input node is the following

```
Input-node          ::  [PRIORITY]
                        Signal-identifier
```

[*Variable-identifier*]*

[*Provided-expression*]

[*On-exception*]

Transition

This grammar does not state how a programmer can specify a concrete input node. Nevertheless, this defines that an input node definition must define these attributes. Roughly speaking, the input node expression must specify, there may be an optional *PRIORITY* quotation provided, a mandatory signal name with optional variables, an optional provided expression and an exception handling expression and a mandatory transition. All of these composite objects, except for the quotation object, resolve in further composite objects. Hence, this results in an abstract syntax tree with the complete SDL system specification object as the root node.

3.4.3 Architecture Description

The architecture describes the static structure of a system. An SDL-2000 system consists of several *agents* representing logical entities [FHL+00]. They constitute the fundamental specification concept of SDL-2000. Agents may group several other agents together and can execute a behavior. Their behavior can be observed. Agents describe active components on the observed system and can also act as a container for other agents. SDL is aware of few types of agents: *system*, *block* and *process* where a system is a specialized form of a block.

The set of agent instances is defined by agent type definitions. Interaction diagrams provide a specific view on the structural composition of the agent instances and details the communication infrastructure between them. In general, an agent type definition can be split into three parts: First, the static structure that is irrespective of time by an interaction diagram that provides insights of the internal structure and composition of the agent. In particular, this depicts the contained agents, the internal communication infrastructure by means of channels and the outside view on the agent consisting of interaction points (gates) and interfaces supported or required at those gates. Second, the behavior specification given by the agent's state machine which is an extended finite and communicating state machine. Third, the internal data types and variables of the agent.

The *system* is the highest level of abstraction in the overall structure. Everything outside the system is considered as the *environment*. It is assumed that the environment acts in an agent-like behavior and structure because a communication between a system and its environment is possible. A *block* can be used to split a system into smaller, encapsulated parts which can help to improve intelligibility, readability and manageability of the overall system specification. *Blocks* are mainly for logical grouping and do not necessarily have a physical correspondence or impact on the target platform. Blocks can contain other blocks and processes which are executed concurrently. *Processes* describe the behavior of the system and therefore, are the active entities in a system. A process may contain further processes, but must not contain any block. Multiple processes within a process are executed in an interleaved manner. In particular, only a single transition is executed at a time. Further structural features are *packages* which contain declarations and definitions like a system, but packages do not define their own scope. A package can be imported to a system or into another package and thus enabling re-usage of predefined objects.

SDL has a complete object-oriented design concept. It covers the main concepts of identity, classification, encapsulation, polymorphism and inheritance. The terminology in SDL varies from some object-orientated concepts, e.g. an object-orientated class is a type in SDL and objects are called instances in SDL. All agents can be classified in types: system type, block type and process type. As all these types can also be defined in system level, they also have the same scope like the system. This

differs from object-oriented concepts with strict scoping rules. A type specification has to be instantiated before it can be used. In most cases, one or more block or process agents are specified in a package or at system level before they are instantiated in the target system. Using a block or process instances without explicitly specifying a type in a structural description yields an implicit type declaration. The architectural modeling elements in SDL-2000 are shown in Figure 9.

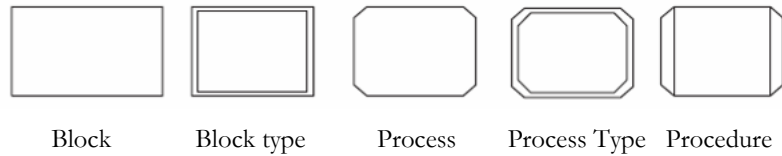


Figure 9: SDL Architectural Modeling Elements

The following Figure 10 gives an SDL system overview. The system consists of a block agent *blockA* and an instance *blockB* of the block type *blocktypeB*. *blockA* is partitioned into two processes *procA* and *procB*. An instance of the block type *blocktypeB* named *blockB*. As shown in the block type definition, *blockB* contains one process named *proc3*. There are several channels shown in this figure.

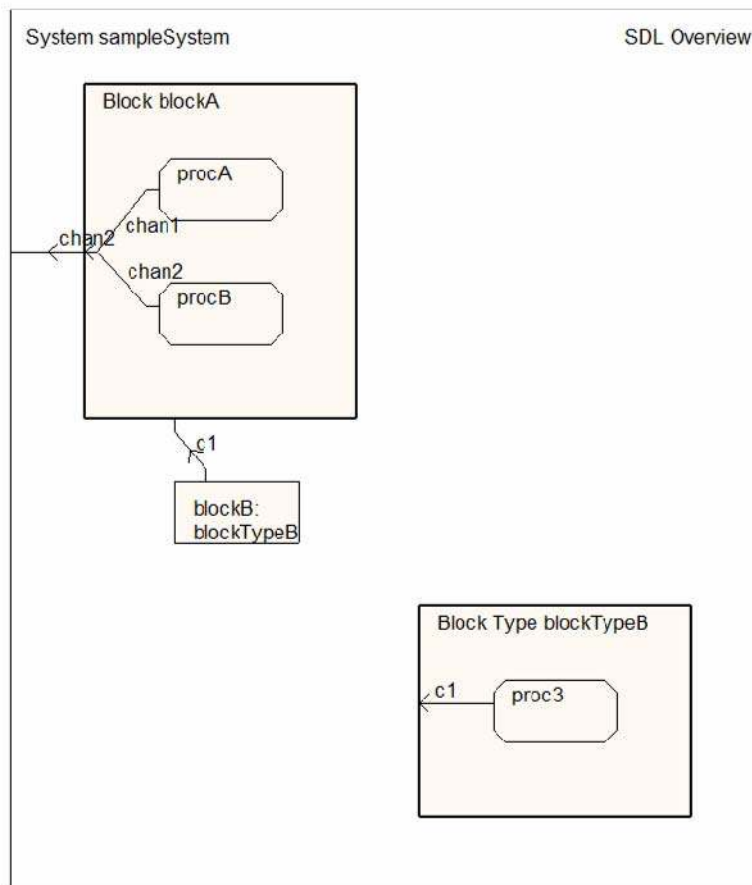


Figure 10: An SDL System Definition

3.4.4 Behavior Description

A behavior is the description of the dynamic structure of a system. Behavior is described by means of process agents. Each agent has an own identification and an infinite input queue for stimuli. The lifetime of an agent is independent from other agents. Processes are defined in the static description and are instantiated from this definition when the system is executing. Multiple agent instances can

exist during the run-time execution of the system at the same time. A special notation allows specification of the initial and maximum amounts of instances of a process agent in the architectural description.

The behavior of a process is described by means of a communicating *extended finite state machine* (EFSM). An EFSM is a tuple of a non-empty set of states, a set of inputs, a set of outputs, a set of transitions, an initial state and a domain set tuple of variables. SDL provides a set of several model elements that allow the specification of an EFSM which are shown in Figure 11, taken from [Obe01].

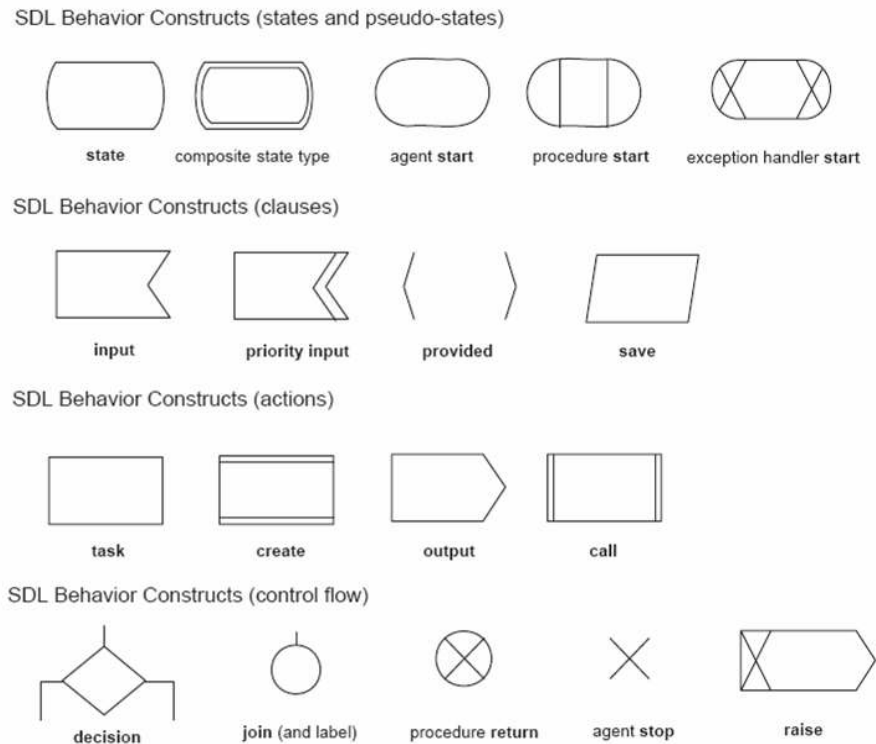


Figure 11: Graphical Notation Elements in SDL-2000

The main entities of a state machine are states. When the state machine goes from one state into another state, it is called a *transition*. A transition is a directed link between two states and is only taken when a specific stimulus is received. The following Figure 12 shows a process defined by means of SDL EFSM modeling elements: The start of the execution begins when the process agent starts. An initial stimulus will occur that is sent to all state machines which have been instantiated. The process will now select the outgoing transition to the state named *state1*. The process waits in this state until a stimulus is received. In SDL, a stimulus is called a *signal*. Mainly, a state can only be left by a triggered transition if a signal is received. As shown in the example, if the signal named *sig1* is received or the signal *sig2* is received, the state *state1* is left and the corresponding transition is selected. When a transition is taken, actions can be executed. For example, if a *sig1* signal is received, task *task1* is executed (the instructions are not shown) and the signal *sig4* is sent out. Then the procedure *call1* is called and after the procedure has been returned the process goes into state *state2*. Sending a signal is a signal output. Outputs are used to send signals that possibly contain values as parameters to other processes. Thus, it provides a mechanism to communicate with other processes. Additionally, the distinct destination of the signal can be specified by means of *to* or the outgoing gate or conveying channel can be specified by *via*. The call construct is used to call a procedure with an optional list of actual parameters. In addition, decisions are supported which implement a dynamic conditional branch of the control flow of a process. A process can end its lifetime by means of a process termination construct. After a process has been terminated, it does not perform any action

anymore. It can only be re-started by a new instantiation after system start. Continuous signals are stimuli that are received if a condition evaluates to a Boolean true value.

Each agent has an infinite first-in first-out (FIFO) input queue for signals. Signals are successively retrieved from that input queue until a signal can trigger a transition, called *consumption* of a signal in SDL. Signals which cannot trigger a transition are discarded unless they are explicitly saved. This is possible by means of the save symbol which may specify a set of signals that must not be discarded in this state.

With SDL-2000, the support for exceptions has been introduced. Exceptions enable the handling of errors and unexpected situations or conditions. Exceptions can be raised implicitly or explicitly (by using the *raise* construct) and the execution is branched to an exception handler. The exception handling procedure is specified similar to a procedure definition.

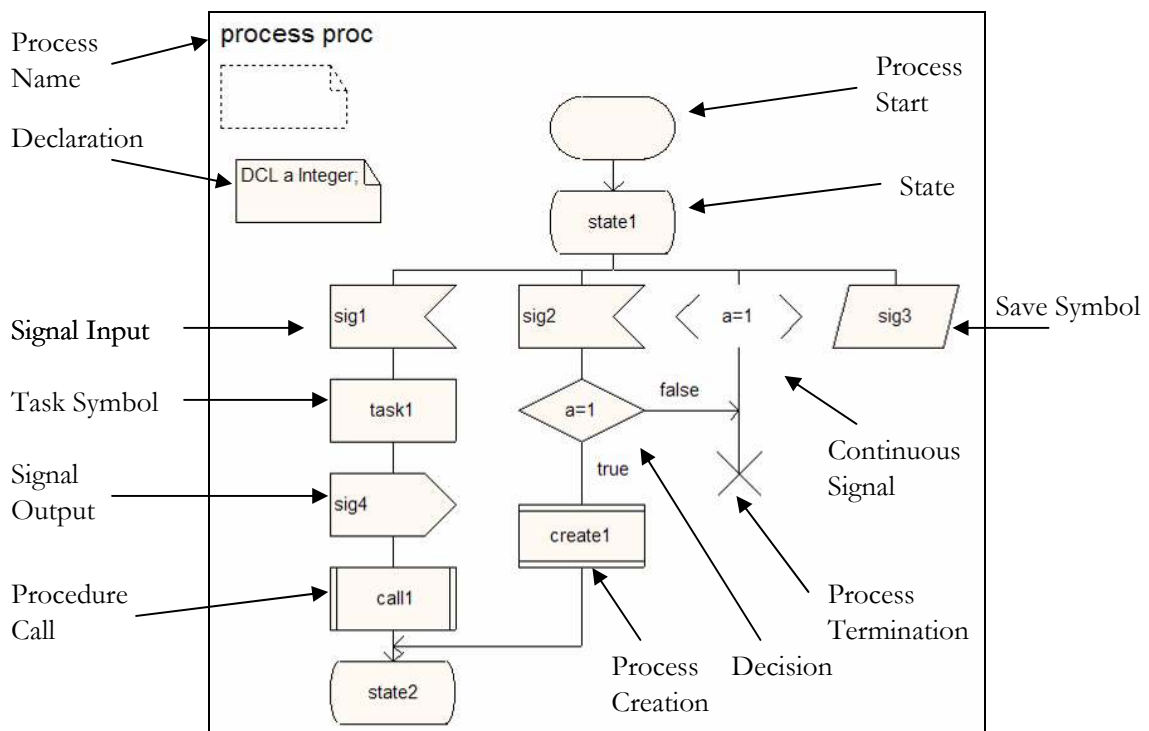


Figure 12: SDL Process Description Constructs

3.4.5 Communication

Process agents communicate by means of *signals*. Communication in SDL is done asynchronously which means that the sending process continues without waiting for an acknowledgment from the destination. Synchronous communication is possible by remote procedure calls (RPC) in a similar concept as client-server roles.

Signals are defined in a declaration symbol. All declarations are implicitly visible to all owned agents. *Channels* define the communication paths through which agents can communicate with other agents. Signals can traverse a channel according to its arrowhead direction and in both directions. Channels require a signal list of the signals that are allowed to convey the channel in the corresponding direction. Channels can be specified as delaying or non-delaying.

The endpoint of a channel is a *gate*. Channels that connect to an implicitly typed agent instance can implicitly specify gates. Gates have to be specified explicitly if a channel connects to an instance of an agent type. A gate is the interaction point of an agent with its environment. Following Figure 13 depicts the available graphical elements. Channels can also originate from or terminate at the system's

environment. The communication is always reliable (e.g. no signal losses or errors) and signals arrive in the same order as they have been sent out.

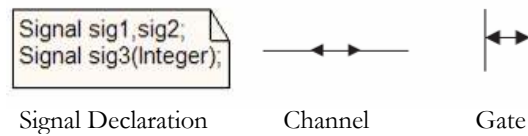


Figure 13: SDL Constructs for Communication

3.4.6 Data

A process can use data values stored in variables. Variables are declared by the keyword *DCL* followed by the identification of the variable and its type. Data types can be declared in any agent and are visible to all owned agents. Information can be exchanged between agents by means of parameter passing as attributes on signals. SDL is platform independent, so it provides a predefined set of data types and allows specification of new ones. Prior to SDL-2000, data types could be specified by means of abstract data types which consist of a data type signature (sort and operands) and its implementation (axioms).

SDL-2000 distinguishes between value types and object types of data types. Value types are always referenced by their values, while object types only provide their reference on their values. Additionally, composite data types like structure and union are available. In more detail, the data model of SDL-2000 is covered in Section 7.7.

3.5 The Message Sequence Charts

The Message Sequence Chart (MSC) is a graphical and textual specification language standardized by the International Telecommunication Union as Recommendation Z.120 [ITU01]. The main purpose of MSC is to visualize and describe the communicating behavior between system instances and their environment [GRS01]. By means of MSC, it is possible to describe the desired (partial) behavior of a distributed system and to describe the concrete behavior that can be observed during simulation or testing. Combined with SDL and the Testing and Test Control Notation (TTCN-3, standardized in the Recommendation series Z.140 [ITU03]), MSC aims to support the design, the simulation, the testing and the documentation stages during the development of distributed communication-based software. In this thesis, MSCs are used to show a specific simulation path of an SDL specification.

The most important elements in MSCs are instances and messages. Instances can be compared to SDL processes or blocks and can exchange messages asynchronously with other instances or with the system environment. In its graphical representation phrase instances are depicted as vertically aligned lines, possibly with a labeled header and a finalizing instance end which specifies the end of the MSC (not necessarily coincident with the termination of the instance). A message is represented by a directed arrow and can have a label which assigns a name to the message. Optionally, parameters of the message can be specified. The source of the message line denotes the occurrence of message sent; the arrow part denotes the consumption of the message. The boundary of the diagram specifies the system's environment. An instance is also able to send messages to the environment and is able to receive messages from there.

Instances can perform actions during their lifetime. This is depicted by a rectangular box on the instance axis. The condition in which an instance currently is in can be specified by using a hexagon.

A condition can roughly be compared to an SDL state, but its main purpose is to decompose an MSC along the timeline. Another construct is the specification of timers and timeouts. A timer start can be specified by an hourglass symbol and its corresponding timeout can be depicted by a message reception originating from the timer. An overview of the available constructs in MSC is shown in the following Figure 14. Most parts of the MSC standard are now integrated in the Unified Modeling Language 2 which is covered in more detail in the following Chapter 4.

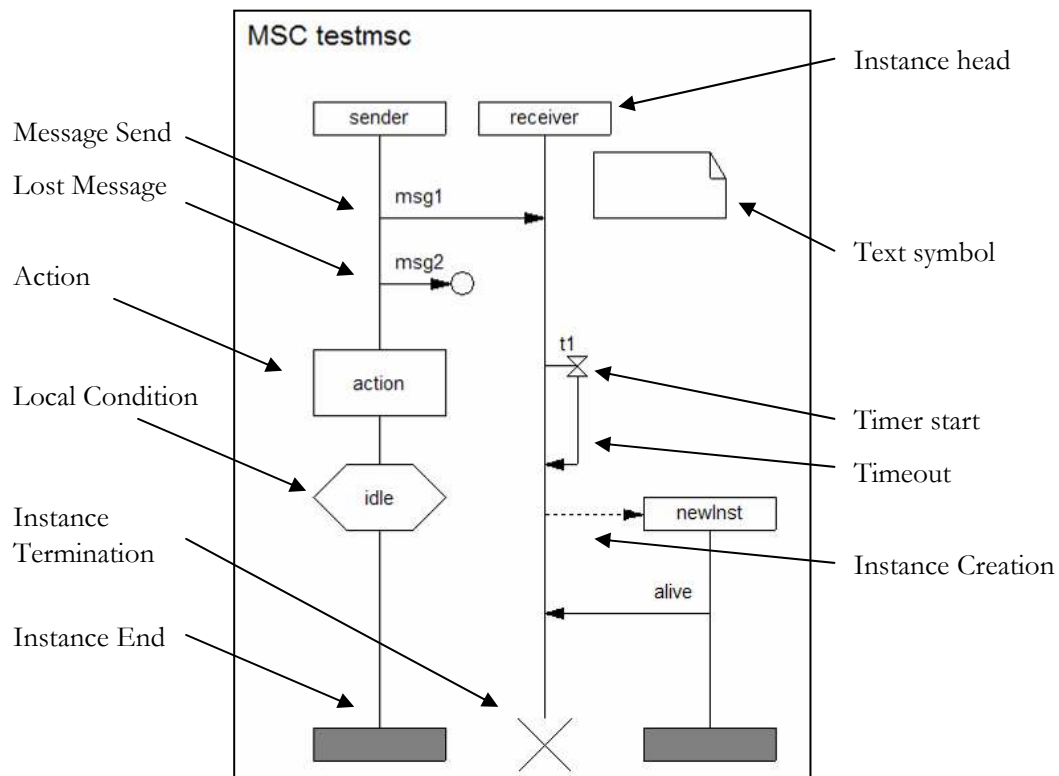


Figure 14: Basic Language Constructs in MSC

3.6 Summary

This chapter has introduced the fundamentals of description techniques for communication protocol engineering. In the first section, the development process for services and protocols has been described. This included the specification and the description stages including the requirements for each of them. The second section has described the model-based development for communication protocols and has given an overview on the mathematical formalism used for this purpose. The third section has outlined the benefits of using formal description techniques for protocol engineering. It has enumerated the components a formal description technique comprises. This included the definition of a formal syntax and a formal semantics. Both concepts have been introduced briefly.

In the fourth section, a concrete formal description technique has been presented, namely the Specification and Description Technique (SDL). The language with its architectural and behavioral specification concepts has been explained and some graphical notations have been shown. The language concept of SDL is used as the conceptual basis for this profile described in this thesis. Another formal description technique has been described in the fifth section, namely the Message Sequence Charts (MSC). A MSC is a formal method to visualize and describe the communicating behavior between instances of a system and their environment. MSCs are often used in conjunction with SDL during system simulation to visualize the communication between agents.

4 The Unified Modeling Language

Today, the *Unified Modeling Language (UML)* is a widespread notation for analysis and design of software systems. It was created and has been maintained by the *Object Management Group (OMG)*. The UML is used for modeling, visualization, documentation, specification and description of complex software systems and other areas. It is used independently of the application or domain. It delivers notational elements for the static and for the dynamic aspects of analysis, design and architecture. In particular, the UML supports the object-orientated paradigm. The UML has its strengths at higher architectural or conceptual levels. It is used for modeling hardware, business processes, structures and systems engineering.

4.1 History of UML

The first object-oriented modeling languages started to appear between the middle of 1970 and the end of 1980. During this period, a number of methodologists published different ideas and approaches to object-oriented analysis and design. Thus, the amount of identified modeling languages increased from a handful to more than fifty. This process culminated in the early 1990's which are often referred to as the years with *the war of the methods*. With the beginning of this decade a high amount of disparate object-orientated methods and modeling techniques were available for software engineers. These methods covered a broad spectrum ranging from the *Object Modeling Technique (OMT)* developed by Rumbaugh to Booch's *Object Oriented Design (OOD)*, over to Jacobson's *Object-Oriented Software Engineering (OOSE)* and to *Object Oriented Analysis and Design (OOA&D)* by Martin and Odell. Different visualization styles, methodologies and design targets combined with inconsistencies and incompatibilities hampered a coordinated workflow and communication between system engineers. Figure 15, taken from [RHQ+05], gives an overview of the variety of methods available at that time and the evolution of methods.

One of the major practical problems was that the potential user of an object-orientated method became unsure which method would best satisfy his requirements and would be best for the goals to achieve. This uncertainty discouraged many users and companies from using such a method for a project. So, they remained using the old conventional methods of analysis and design.

Consequently, new iterations of these methods began to appear which expanded and incorporated previous modeling techniques. These new methods were not developed from scratch, but by combining several useful approaches and features from existing ones. Many methods became extinct while a few clear prominent methods emerged at the end of this process. Mainly one method evolved from this period: The *Unified Modeling Language* created from the *Unified Method* (Booch and Rumbaugh) and the approach used in OOSE by Jacobson. One of the pioneering companies at this time was the *Rational Software Corporation* (now part of the IBM Corporation).

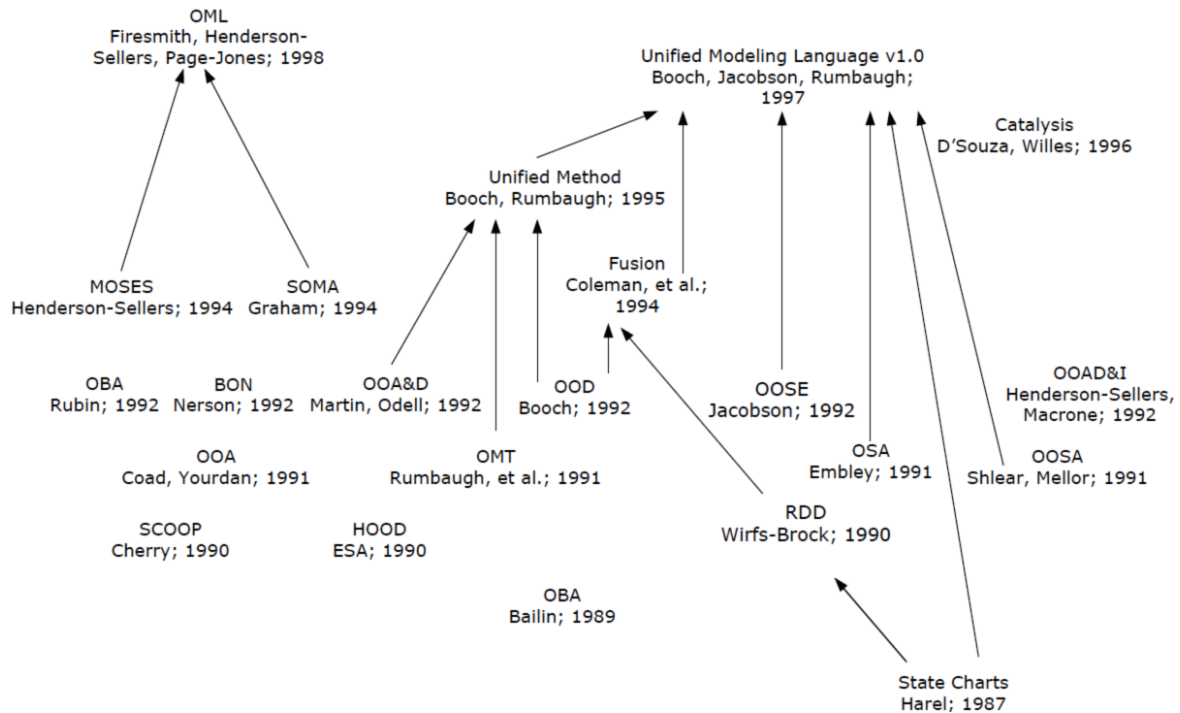


Figure 15: From Method Wars to the UML

UML was further unified by Booch, Rumbaugh and Jacobson (so-called *three amigos*) until the UML Versions 0.9 and 0.91 were released. They adopted four goals [BM98]:

- to provide a notation that enables several views on the same system (instead of only capturing portions of the system) using object-oriented concepts,
- to relate between the concepts and the implementation of a system,
- to consider the scaling factors that are inherent to complex and critical systems,
- to define a modeling language that is comprehensible by both humans and machines.

This version was especially adapted for the needs of the industry. From this point, several companies joined the process which resulted in the UML Version 1.0. New requirements influenced further refinement and development. For example, IBM promoted the integration of the *Object Constraint Language* (OCL) into the UML. OCL is a language framework to ensure consistency by specifying modeling related conditions which are guaranteed to be satisfied. OCL was integrated into the UML with the release of Version 1.1. This was the first version evolved from the OMG joint work as a standardized framework for object-oriented modeling.

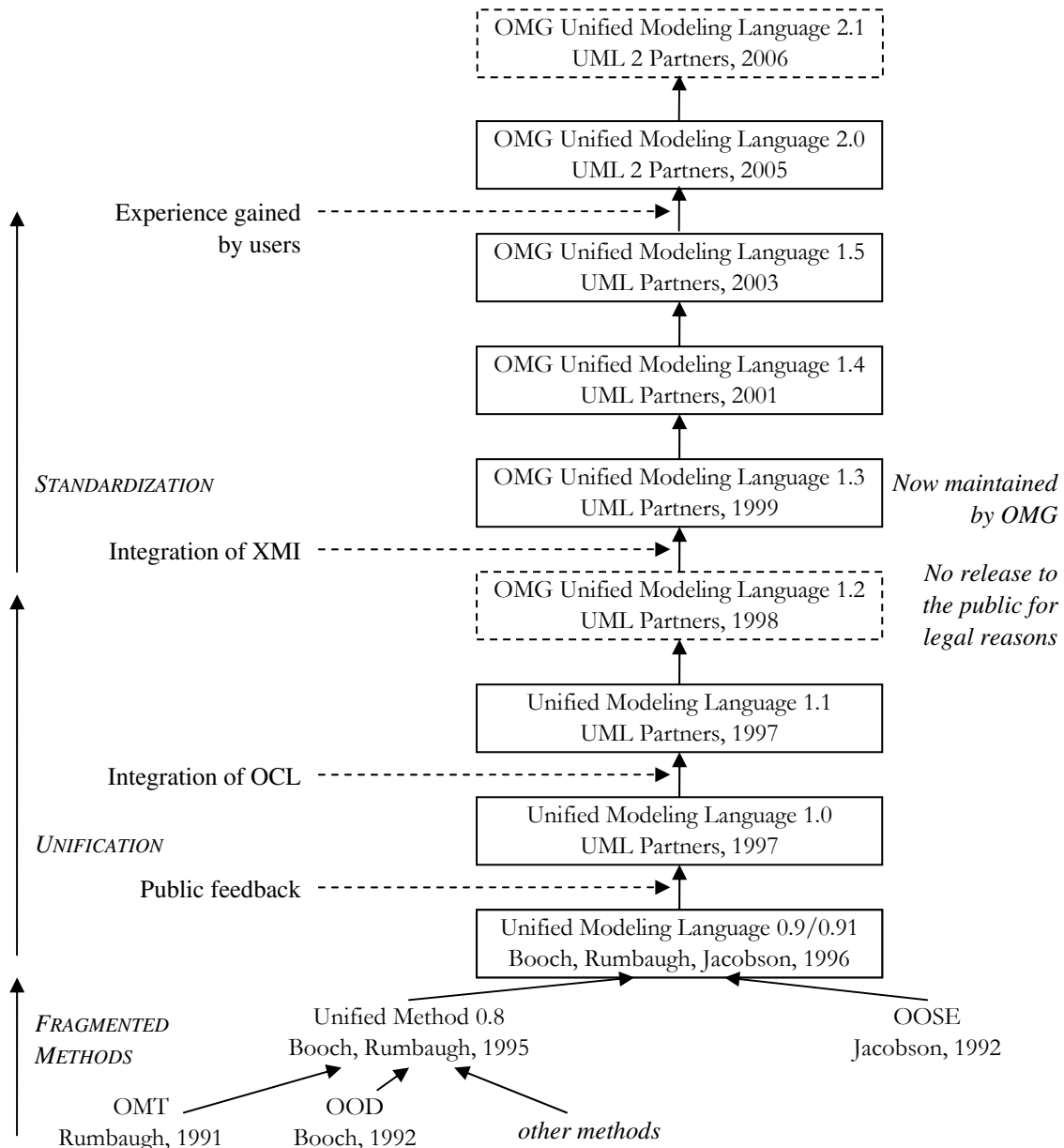


Figure 16: Evolution of the UML

In 1999, the UML integrated the *XML Metadata Interchange Format (XMI)*. A standard based on the *Extensible Markup Language (XML)* [BPS+06] for exchanging metadata information whose metamodel can be expressed in Meta-Object Facility (MOF). This enriched the UML from a solely graphical notation with a textual notation to a notation with same cardinality. XMI is discussed in more detail in the Section 4.6.

The experience gained by the users continued to influence recent versions and finally led to the launch of the UML Version 2.0 in 2005 continuing with the current working revision 2.1 launched at the beginning of 2006. Figure 16 illustrates the evolution process of the UML from its roots to its current version.

4.2 The UML Version 2

There are several reasons that have driven the development of subsequent releases of the UML. The most important impacts were complaints about its complexity and hugeness. Furthermore, new software engineering techniques introduced new requirements to a description method. For instance, software models were needed for the description of technical systems with hard time bounds, known today as *real-time* or *embedded* systems, or component-based development techniques (which are used by the *Java Platform, Enterprise Edition* – abbreviated with *Java EE* or *J2EE*, for instance).

Hence, the first steps for an update of UML were attempts to reduce its complexity. After several discussions the results of a streamlining attempt focused on the re-build of the UML metamodel from scratch with a coherent and exhaustive usage of the OCL. Additionally, the reusability of basic constructs whenever possible was enforced. Relations between static and dynamic types of diagrams were improved to allow a better executability. Parts with a weak semantic description were clarified. Encapsulation and scalability in dynamic diagrams were better supported. Old restrictions in activity modeling were removed and support for hierarchical modeling of a system (ability to decompose a system into smaller parts) was added.

This overall process led to two different UML 2 documents that extend and reference each other: The Infrastructure [OMG04a] and the Superstructure [OMG05a] documents. The Infrastructure document describes basic language constructs and the architectural foundations. Based on this, the Superstructure document describes the diagram notation and its semantics. Both documents were accepted at the end of 2003 but – due to delays – the Superstructure document has been finalized in the mid of 2005 whilst the Infrastructure document is still under review. At the time of writing (mid-2006), the UML 2.1 [OMG06] and UML 2.2 are already underway. Both revisions are primarily maintenance and error corrections.

4.2.1 UML 2 Diagrams Types

The UML 2 offers a total of thirteen different diagram types. Six of them provide elements and notations for the modeling and description of static part of a system. These are the *structure diagrams*. The remaining seven diagram types describe the dynamic part of a system. These are the *behavior diagrams*. The behavior diagrams are further categorized into *interaction diagrams*. With the UML 2, the following diagram types have been introduced. They were not part of the UML in earlier versions:

- the *timing diagram*,
- the *package diagram*,
- the *interaction overview diagram* and
- the *composite structure diagram*.

Figure 17 gives an overview of all diagram types that are available in the UML 2. The UML standard defines several elements with different semantics and size for each aspect. These elements are grouped into these diagram types. For example, the element group consisting of classes, attributes, operations and associations represents the basic class structure. States combined with transitions and events define the dynamic state changes in a system. However, the UML standard is very unconstrained with the usage of elements and diagrams. Elements can be used in almost any diagram type if it is desired. Of course, in most cases this is not helpful or even makes no sense.

DIAGRAM TYPES OF THE UML 2		
STRUCTURE DIAGRAMS	BEHAVIOR DIAGRAMS	
Class Diagram		Interaction Diagrams
Package Diagram	Use Case Diagram	Sequence Diagram
Object Diagram	Activity Diagram	Communication Diagram
Component Diagram	State Machine Diagram	Timing Diagram
Composite Structure Diagram		Interaction Overview Diagram
Deployment Diagram		

Figure 17: Overview of UML 2 Diagram Types

Related to language and information theory UML modeling elements do not have a defined, fixed representation, but only have to comply with the rules of the UML metamodel. This metamodel specifies the abstract syntax of any UML model element. Figure 18 shows an example how UML elements can be visualized in different representation styles, taken from [RHQ+05]. The most common notation is the graphical notation. The *XML Metadata Interchange (XMI)* notation is merely used as a non-proprietary diagram interchange file format between different UML tools. The Human-usable Textual Notation (HUTN) [OMG04b] is currently rarely used and poorly supported. As its name implies, it is a concrete, textual notation for UML models in a human-understandable format.

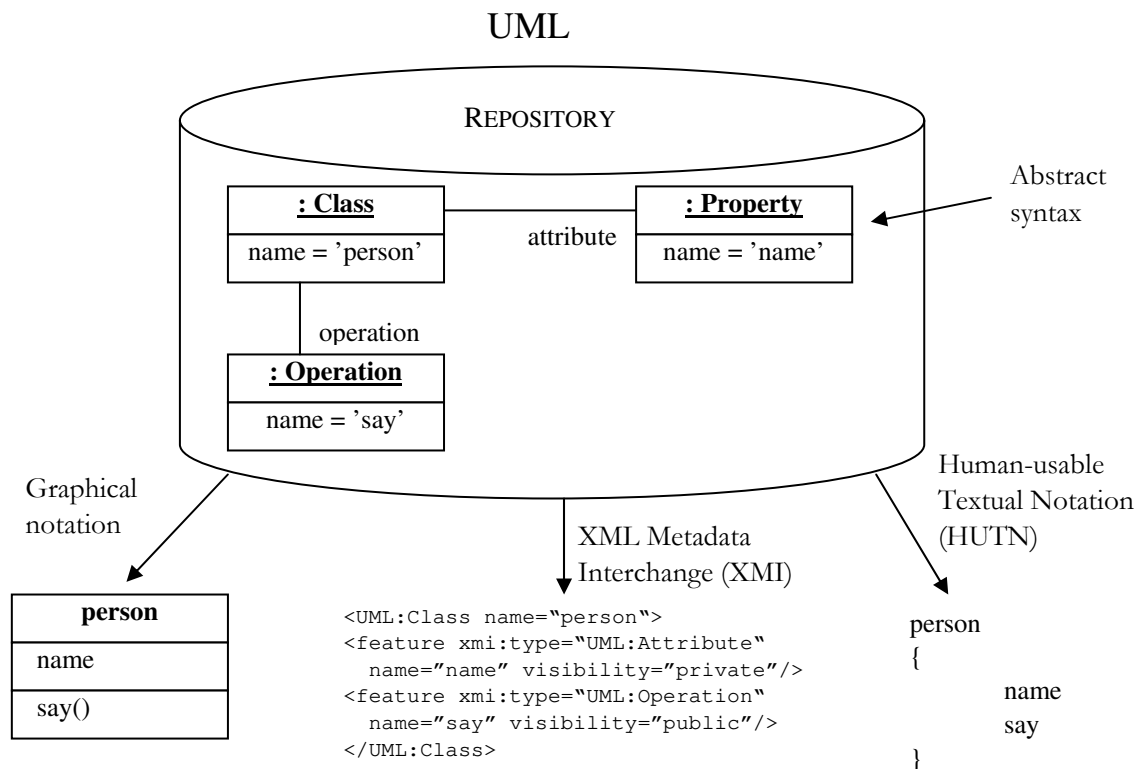


Figure 18: UML Repository and Representations

UML modeling elements are depicted within a diagram. This diagram can feature a frame header which allows determining the type, name and possible parameters of the diagram. The UML allows hierarchical model structures as well as (multiple) referencing, decomposing, the construction of

namespaces (e.g. by the use of *packages*) and ownerships between elements (e.g. a class has a behavior modeled by a state machine). Therefore, the frame header clarifies where this diagram belongs to and which aspect of a system is depicted. This is especially required as UML supports multiple viewpoints. Multiple viewpoints allow defining multiple diagrams of the same UML element showing it in a different manner with a focus on different aspects or abstraction levels.

4.2.2 Structure Diagrams

In the following, some of the relevant structure diagrams for the UML CS profile are briefly introduced: Classes and Composite Structures. Structure diagrams are a type of diagrams that shows the elements of a specification which are irrespective of time.

Class diagrams are used to describe the types of objects in a system including their relationships to each other. Class diagrams visualize structure, relationships and internals of classes by using a set of design elements such as classes, packages and objects. When designing a system they describe three different perspectives: conceptual, specification and implementation. During the creation of diagrams, each of these perspectives can be used to consolidate the overall design. Class diagrams only describe what interacts but not the behavior when instances do interact; class diagrams are static. An *object* is any person, concept, thing, event, report or machine applicable to the system. Objects can own attributes and have methods. Objects are the instances from a class definition forming a type/instance dichotomy. Classes are the main building entities of an object-oriented application and are typically modeled as rectangles with three sections. These sections are labeled with the name of the class, the attributes of the class and the operations of the class. *Attributes* are stored information of an object while methods describe what an object or class can do. *Methods* are the object-oriented equivalent of functions and procedures. They define the behavior of an *Operation*. Instances of classes (objects) are often related to or referring to other objects. Such relationships are defined by *Associations*. They are drawn as possibly directed lines connecting the classes whose objects are involved in that relationship. An association may also define multiplicities and constraints. Furthermore, similarities often exist between different classes. In many cases, two or more classes share the same attributes and the same methods. The object-orientated concept of inheritance enables re-use of existing data and code easily. The UML notation for inheritance is a line with an unfilled arrowhead pointing from the subclass to the superclass. A *generalization* relationship is used to indicate inheritance. It is drawn from the specific class to a general class; the opposite direction would mean *specialization*. The generalized implication is that the source inherits all the properties and characteristics of the target.

An *interface* is the specification of a behavior that implementers agree to implement when an interface is realized. By realizing an interface, classes are accepting a kind of contract to support a required behavior by providing the implementation of specific operations. A class may signal to require an interface. Such classes can invoke this contracted behavior on realizing classes. An interface is quite similar to a class but with several restrictions: All interface operations are required to be of public visibility, abstract and all interface attributes must be constants. For drawing an interface, the UML provides two different alternative notations: an interface element can be drawn analogously to a class explicitly detailing its specified operations, but labeled with the *interface* keyword. Alternatively, it may be drawn as an unfilled circle without any explicit operation defined. Interfaces which are bound to a specific class can be defined as either *provided* or *required*. A provided interface is the confirmation of the realizing class to provide an implementation of the operations defined by the named interface element. This relationship is defined by a *realization* link between the class and the interface. A required interface is the declaration of a class to require and to be able to communicate with some other class that provides the operations' implementation defined by the interface. A

dependency link depicts this relationship between the class and the interface. An example of a Class diagram is presented in the Figure 19.

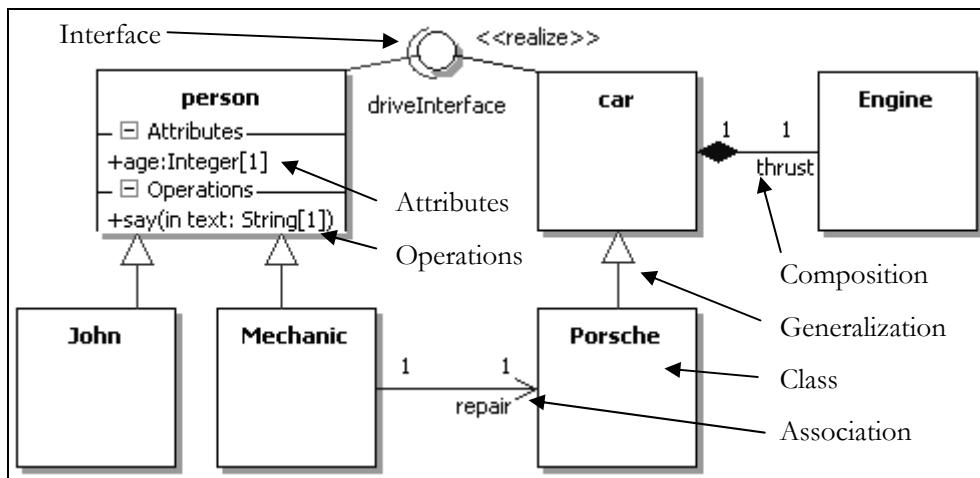


Figure 19: Example of a Class Diagram

Composite structure diagrams show the internal composition and arrangement of a component or a class. The focus lies on the internal structure of a class with respect to its interaction points to other parts of the system. By using a composite structure diagram it can be shown

- what parts the component (e.g. class) consists of,
- how these parts are internally connected and
- the interfaces this component provides or requires.

The several parts inside the component are not specifications of instances, but classes or other components, thus showing the configuration and relationship of parts. A *part* is an element that represents a number of instances which are owned by a containing instance. All parts together perform the emergent behavior of the containing class. A part is drawn as an unfilled rectangle and is contained within the component element or body of a class. A *port* is a typed element that represents an externally accessible part of an instance of a contained class. Ports define the interaction and interaction points between an instance and its environment. In particular, a port specifies the services that a class provides and the services that it requires to be provided from its environment. A port is shown as a labeled rectangle placed on the border of a part, class or composite structure. Figure 20 provides an example of a Composite Structure diagram.

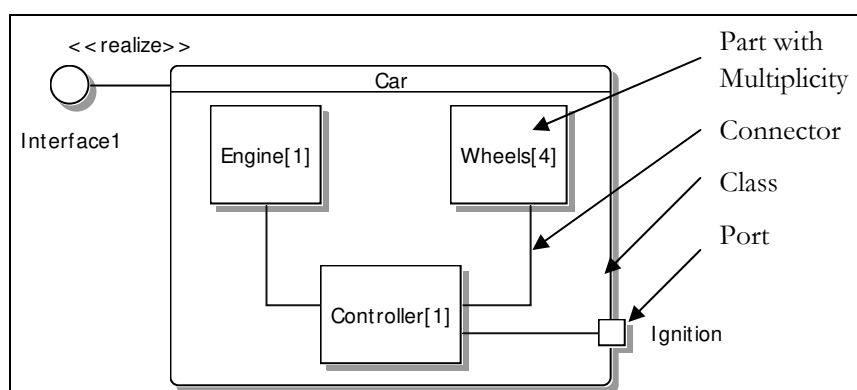


Figure 20: Example of a Composite Structure Diagram

4.2.3 Behavior Diagrams

In this section, the relevant behavior diagrams for the UML CS profile are introduced briefly: States and Activity Diagrams. Behavior diagrams are a type of diagrams that depict behavioral features of a system. This includes activity, state machine and use case diagrams as well as the four interaction diagrams which are not described here. For a more detailed description of the available UML 2 diagram types refer to [RHQ+05].

State diagrams focus on the behavior of an object that is executed by a sequence of events. This feature is especially useful for modeling reactive systems. A state diagram visualizes the flow of control that is caused by event-triggered transitions which lead from one state to another state. It describes the possible timely sequence of states and actions through which an active object can go during its lifetime as a result of its reaction to discrete events. A state diagram describes a finite state machine. It consists of:

- a finite, not-empty set of states,
- a finite, not-empty set of events,
- state transitions,
- an initial state and
- a set of terminate states.

A *state* reflects a situation during the lifetime of an object in which this object performs some actions or waits for some event to occur. According to the UML metamodel, states can belong to one of the following categories: *simple states*, *composite states* and *submachine states*. In contrast to simple states, composite and submachine states comprise of further sub-states. In addition, there are final states and so-called *Pseudostates* (initial, deepHistory, shallowHistory, join, fork, choice and junction) defined.

Transitions are directed relationships between two states. They indicate that an object in the source state will enter the target state and perform specific actions, but only when a specified event occurs and certain conditions are satisfied. A transition may have a *trigger*, a *guard* and an *effect*. A trigger is the cause of the transition to execute. Triggers can be invoked by a signal, an event, a change in some condition or a passage of time. Guard is a condition which must be satisfied in order to enable the trigger to cause the transition. Effect is a set of actions that will be invoked on the object that owns the state machine resulting from the triggered transition. When a trigger occurs that cannot trigger any transition, a state may specify it to be deferred. This trigger will then remain pending. This will last until a state is reached where it can trigger a transition or is deferred again.

Entering a *terminate* pseudostate indicates that the lifetime of the state machine has ended. A *choice* pseudostate is a dynamic conditional branch in the control flow of a state machine. *Junction* Pseudostates are used to chain together multiple transitions. Junctions that split an incoming transition into multiple outgoing transitions realize a static conditional branch. A *history* pseudostate is used to remember the previous state of the state machine. The execution semantics of a UML state machine is based on the *run-to-completion* (RTC) principle: Exactly one event is processed at a time. When the machine is in a stable configuration, the subsequent event is processed as soon as all consequences of the previous event have been implemented. Therefore, an event is never processed when the state machine is in some intermediate or unstable situation where further events are pending. An example of a State Machine diagram is given in the following Figure 21.

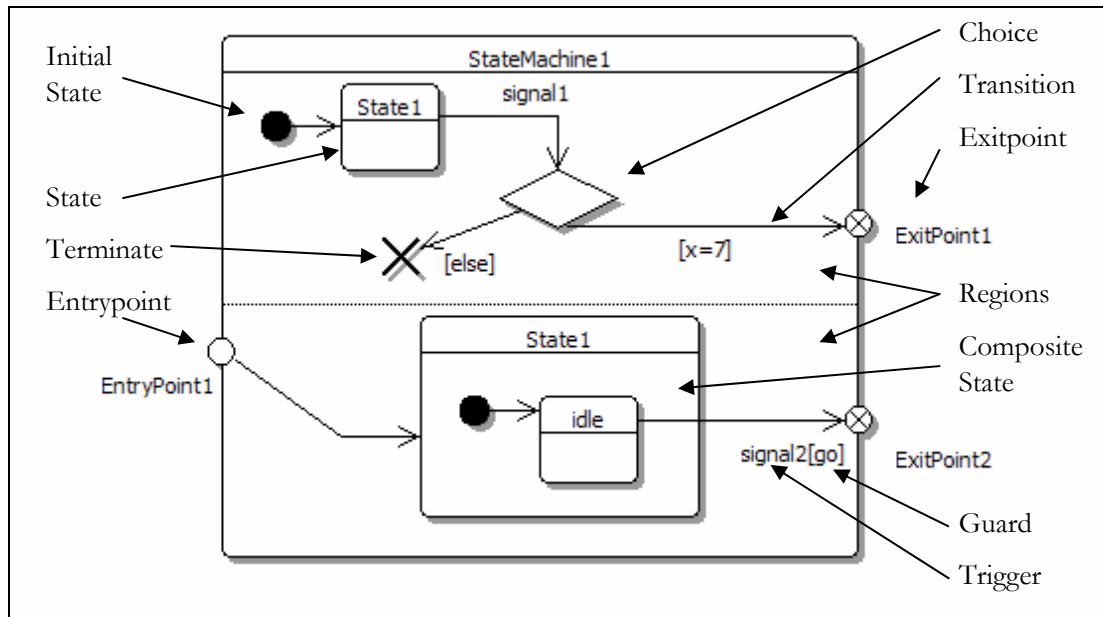


Figure 21: Example of a State Machine Diagram

Activity diagrams are used to visualize the sequence of activities, i.e. a possible behavior of a system. Activity diagrams show the behavior or workflow from an initial vertex to a final vertex with respect to the intimate details of the control and data paths that exist through the execution of behavior. They may be used to explain and describe situations where concurrent processing may occur. An activity is the specification of a parameterized sequence of behavior. It depicts the flow of control, by means of *tokens*, and data values along the control and data edges. Activity nodes can operate on and route them to other nodes or they can even store them temporarily.

There are the following main kinds of nodes in an activity diagram: *Action nodes* (Actions) represent single steps within an activity. They operate on the control and data values they receive and provide control and data to other actions. Such nodes are distinguished in several types: *Control nodes* route control and data tokens through the diagram. *Decision nodes* and *merge nodes* are used to split or join control nodes based on specific criteria. The control flows originating from a decision node have guard conditions which allow to steer the flow if the guard condition is met. *Initial* or *start nodes* define the beginning of activities. There are two types of final nodes: *activity final* and *flow final* nodes. The latter flow final node denotes the end of a single control flow without affecting the remaining control flows whereas the activity final node denotes the end of all control flows within the activity and thus, terminates the activity as a whole. *Forks* and *joins* indicate the start and end vertex of concurrent threads of control. A *join node* contrasts to a *merge node* in that the join synchronizes two inbound flows and produces a single outbound flow. The outbound flow from a join cannot execute until all inbound flows have been received. Therefore, a merge forwards any control token that it receives straight through it. *Object nodes* can hold data tokens temporarily while they traverse through the activity graph. An object node is an abstract activity node. It is part of the defining object flow in an activity.

Activity nodes are connected by two different kinds of directed activity edges: *Control flows* connect action node. They show the flow of control from one action to the next. Control flow edges connect to subsequent actions, thus indicating that the action at the target end of the edge will start when the source action has finished. Only control tokens can pass along the control edges. *Object flow* edges connect object nodes. They provide inputs to actions, hence only object and data tokens can pass along object flow edges. Control and object flow edges share the same representation. They are only distinguished by usage. Control edges connect actions directly; object flow edges only connect the input and output pins of actions.

Activity diagrams are somewhat similar to state diagrams. Whereas state machines focus on the object's state, activities focus on the *state of behavior*. These diagrams describe the activities by sequence of activities performed. An example of an Activity diagram is shown in the Figure 22.

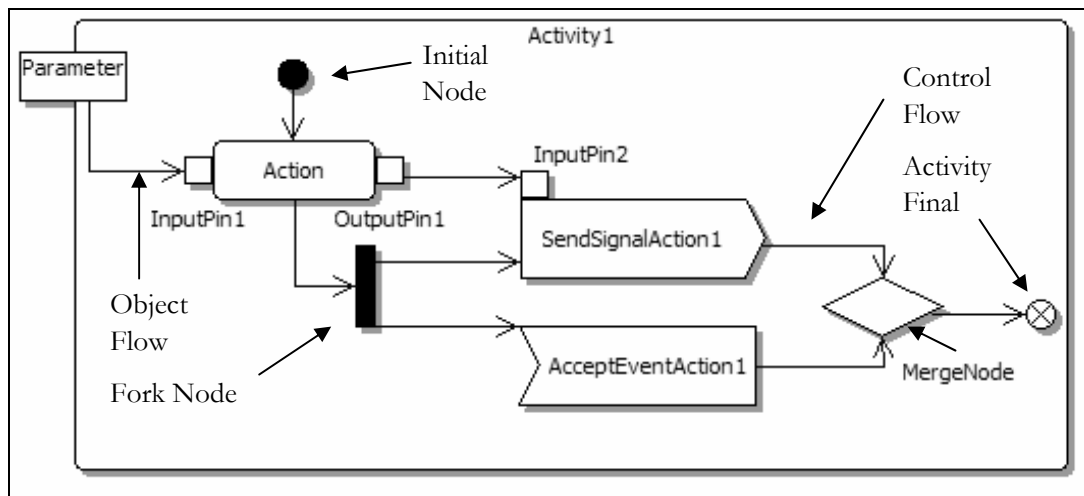


Figure 22: Example of an Activity Diagram

4.3 UML and Metamodeling

The UML is defined by means of the metamodeling concept. Metamodels serve as models for other models [KK02]. Many model elements in the UML represent a type/instance dichotomy. A type represents the essence of an abstraction and the instance forms a concrete sample. In addition, there is a type/class dichotomy in which classes and primitive types implement types.

A type specifies a domain of values and a set of operations applicable to those values. A class implements a type and it provides the representation of attributes and the implementation of operations (methods). This distinction is propagated with subclasses, so that the specification given for a type is valid for all subclasses and a subclass may implement several types.

In other words, modeling describes an information domain by means of a model. This model is described using a specific language – the modeling language. This language contains a set of language constructs with a defined syntax and semantics. Metamodeling describes a modeling language for a model. Hence, to achieve this in a consistent manner a metamodeling language is used for this. It itself contains modeling elements with a defined syntax and semantics.

In essence, metamodeling describes (meta-)models whose domain is another modeling language. For this purpose, the OMG has defined four levels of metamodels shown in Figure 23.

LAYER	DESCRIPTION	EXAMPLE
meta-metamodel (M3)	The infrastructure for metamodeling architecture. Defines the language for specifying metamodels.	<i>MetaClass, MetaAttribute, MetaOperation</i>
metamodel (M2)	An instance of a meta-metamodel. Defines the language for specifying a model.	<i>Class, Attribute, Operation</i>
model (M1)	An instance of a metamodel. Describes the language to describe an information domain.	<i>person, name, say</i>
information (M0)	An instance of a model. Describes the specific information domain.	<i><person_33111>, "john", say()</i>

Figure 23: Four Layer Metamodeling Architecture

The concrete instantiations of the model elements from the meta-meta model layer down to the concrete information layer is pictorially depicted in the following Figure 24.

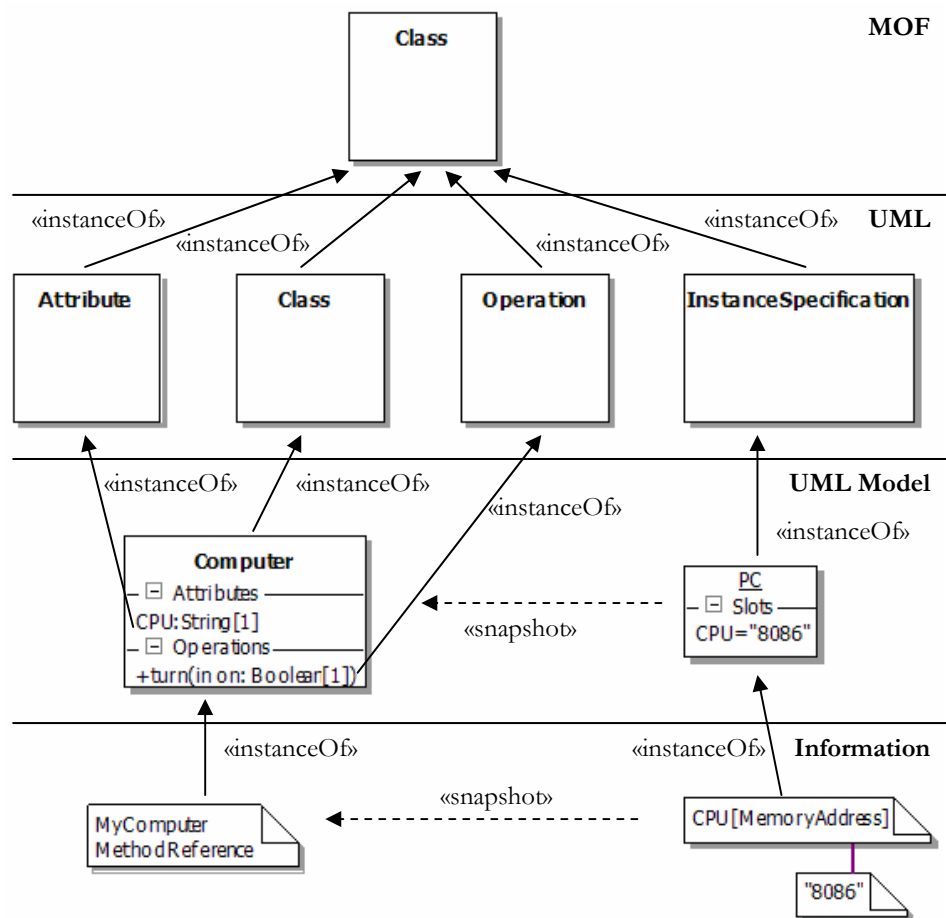


Figure 24: Instance Creation in UML Metamodeling Layers

The information (or runtime) layer *M0* is the lowest layer in the hierarchy. This layer represents the reality and describes the real basis of models. Abstracted from this layer, models are created. This uniformly represents objects and behaviors which are available or will occur at runtime.

The runtime layer is the foundation for the layered design of the metamodeling approach. This real world is not intended to be modeled with all and every aspects. Instead, it is intended to create an abstracted model from this, as models are much easier to cope with than as real-world models. Additionally, many aspects of the real-world objects are irrelevant and the focus lies on essential parts. Consequently, objects of the *M0* layer are instances of the modeling layer *M1*. A model defined in *M1* can be instantiated to a real world object. The metamodel *M2* is the model of a model. For instance, the UML 2 metamodel defines that a class may own one or more attributes and operations. It also defines how many outbound lines may be attached to an activity and so on. It defines the *abstract syntax* of the UML model elements. Figure 25 gives an overview of the UML 2 metamodel.

The meta-metamodeling layer *M3* is the topmost layer in the four-layer-model of the UML. It is the foundation of all metamodeling layers. In most cases, this layer is only of concern for tool developers, metamodeling and code generator engineers. This layer describes what is allowed to be used to build a metamodel, e.g. class, property, generalization, association et cetera. However, it does not define a composite structure diagram, state machine diagram or use case diagram. The meta-metamodel defines a simple class model with exact definitions for multiplicity and is itself defined by an object-oriented class model described in the UML.

The meta-metamodel of UML defined in the *M3* layer is the *Meta-Object Facility* (MOF). The MOF is also an own, independently defined standard by the OMG in [OMG03a].

To summarize, the differences between MOF, MOF-based metamodel, UML metamodel and UML model are that MOF is a language for creating metamodels. Any metamodel created by using MOF constructs is a MOF-based metamodel. The UML metamodel is a metamodel created by using the MOF language, while the UML metamodel itself is a language for creating UML models. The UML metamodel is MOF-based. A UML model is created by using UML metamodel and therefore, it is not MOF-based directly.

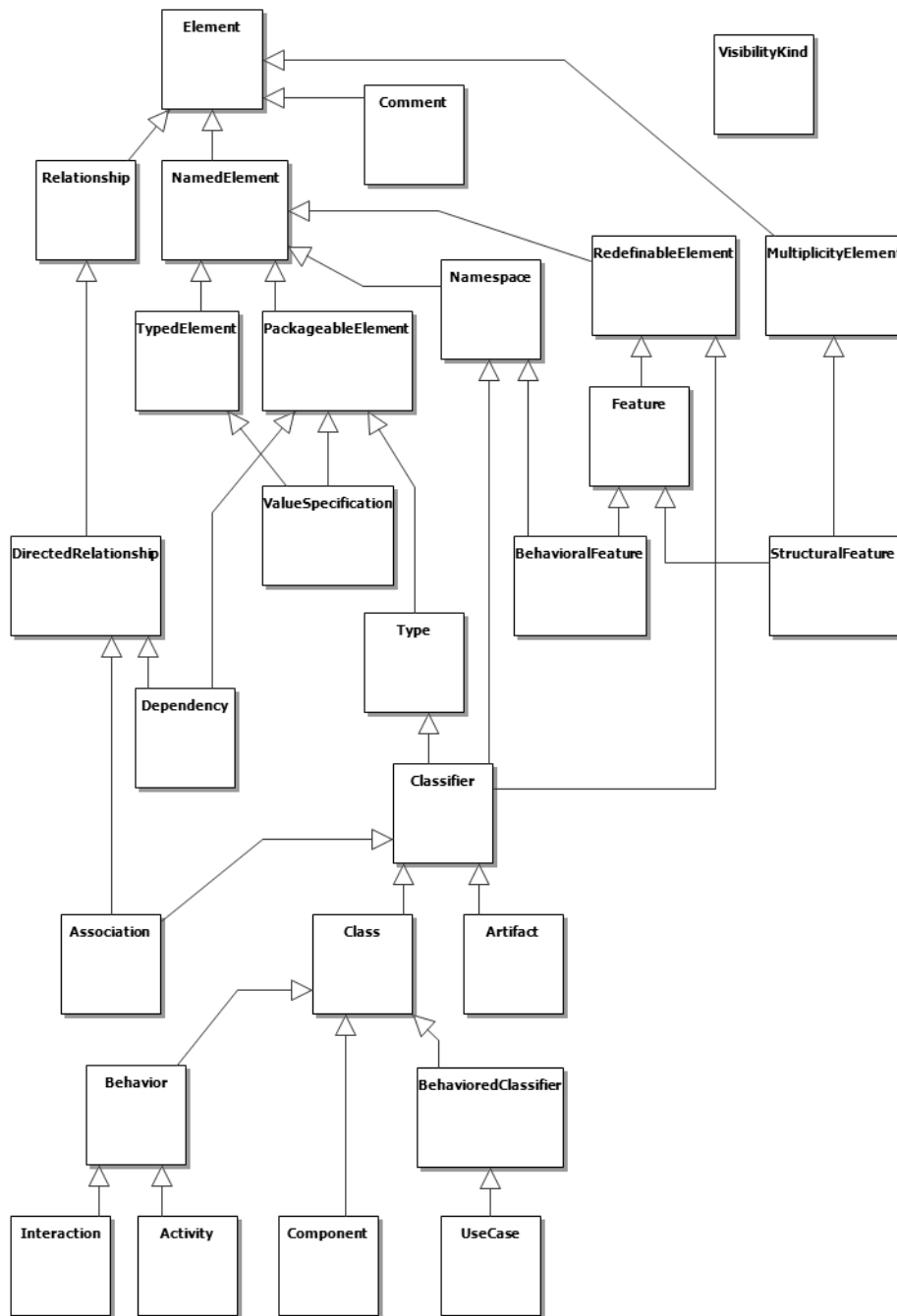


Figure 25: Overview of UML 2 Metamodel Class Hierarchy

The class diagram of the UML metamodel in Figure 25 depicts the basic structure. There can be a few core classes identified. The first one is the *Element* metaclass. Almost all classes in the UML metamodel hierarchy are specializations of this base class. The *NamedElement* metaclass – being a specialization of *Element* – adds an identifier to each instance of a metaclass element. This enables unique addressing of all instance types. The *TypedElement* metaclass introduces the type concept to its specialized metaclasses. It constrains values to a specific range of values. The *RedefinableElement* metaclass provides the basis for object-orientation concepts like overloading and overwriting. Instance definitions of a *RedefinableElement* can be refined by means of further instances. Another important metaclass is the *Classifier*. A *Classifier* is a more general description and is an abstraction of a generic class that is known from object-oriented concepts. In UML, there are 23 different specializations defined from the *Classifier* metaclass.

4.4 UML Extension Mechanisms

The UML has been designed to support a wide variety of application domains. In spite of this design philosophy, it is not possible to cover the needs of every user. In most cases, it is required to apply some adaptations to the UML, possibly caused by the user's methodology or specific requirements of the target system. Especially, new engineering technologies take some time to be integrated into the UML. This concludes that the UML will probably never be optimally suited for the design of any system one can think of.

Profiles have been introduced in the UML 1.3 to enable the extension (or specialization) of the standard modeling language. At that time, there were many requests to put several modeling elements into the UML. This would have evolved into a huge collection of each new software modeling technique to be incorporated into the UML. Profiles were introduced to allow users to add their own touch to the UML. Prior to the UML 2, extending the UML was limited only to the use of stereotypes and profiles. As depicted in Figure 26, there are the following mechanisms available to adapt a metamodel to a specific purpose:

- New metamodel
- Extended and modified UML metamodel
- UML Profiles
- (and not shown) Stereotypes

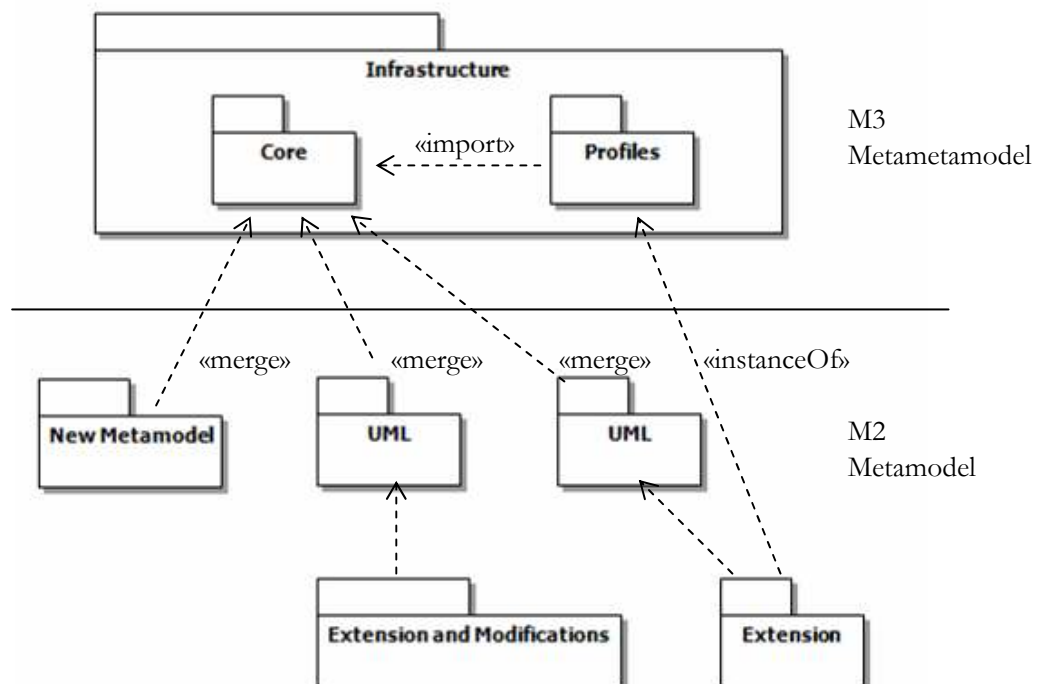


Figure 26: UML Extension Mechanisms

The first option is to define a completely new metamodel based on the M3 layered meta-metamodel. Alternatively, an uncontrolled extension, a modification of the UML metamodel, can be done. This type of extension is called *heavy-weight extension* or *first-class extension*. Controlled extensions based on inherent extension mechanisms are profiles and stereotypes. They are *light-weight extensions*.

4.4.1 UML Profiles

The UML introduces a concept of profiles. This package provides extension mechanisms which allow specialization of metaclasses from existing metamodels to tailor them for different purposes. For instance, this includes the ability to align the UML metamodel for different platforms (such as J2EE or .NET) or domains, such as real-time systems, business processes or systems engineering. It is important to note that the profile's extension mechanism is consistent with the OMG Meta Object Facility (MOF). It is not a first-class extension and therefore, it is not possible to alter existing metamodels. According to the UML 2.1 Superstructure document [OMG06], a profile is

[...] a kind of Package that extends a reference metamodel. The primary extension construct is the Stereotype, which is defined as part of Profiles. A profile introduces several constraints, or restrictions, on ordinary metamodeling through the use of the metaclasses defined in this package.

A profile is a restricted form of a metamodel that must always be related to a reference metamodel, such as UML, as described below. A profile cannot be used without its reference metamodel and defines a limited capability to extend metaclasses of the reference metamodel. The extensions are defined as stereotypes that apply to existing metaclasses.

UML profiles are UML packages with the keyword «*profile*» (with surrounding guillemots). A profile is used to extend a meta-model or another profile. It gives further details and semantics to existing UML elements by providing refined elements that extend existing meta-model classes. For this purpose, a UML profile can consist of *stereotypes*, *constraints* and *tagged values*.

A stereotype is an extension (or specialization) of existing metamodeling elements. It describes the domain specific usage context in which a modeling element (class, relation or package) is placed. Stereotypes are the main components of a UML profile and describe how the UML metamodel can be extended to a specific domain. Stereotypes are specialized classes which are labeled with the keyword «*stereotype*». Model elements can be classified upon instantiation by means of stereotypes. Stereotypes enable the addition of auxiliary UML metaclasses with new meta-attributes and semantics. As all classes, a stereotype can also own properties like attributes which are referred to as *tag definition*. A stereotype can also introduce *tagged values* which assigns an initial value to a tag definition. Tagged values are user-defined and tool- or language-specific name-value pairs and are defined by a name and a type. In contrast to stereotypes, tagged values are used as additional meta-attributes to complement a model element by attaching arbitrary information to a model element.

Stereotypes can also implement generalization hierarchies but only within other stereotype classes. A stereotype extends a metaclass of the metamodel. An extension is visualized by a generalization arrow, but with a filled arrow instead of an unfilled arrow for class generalization. This is a particular association between a metaclass and the stereotype. Both classes are simultaneously instantiated except when the stereotype is marked as being optional (in UML terms by a missing *required* notation) or being abstract. As a stereotype is an extension to a model element, it can be used together with the metaclass model element. For each tag definition of the stereotype, a tagged value can be added. A stereotype can also own its own notation, e.g. by means of a specific icon for a metaclass.

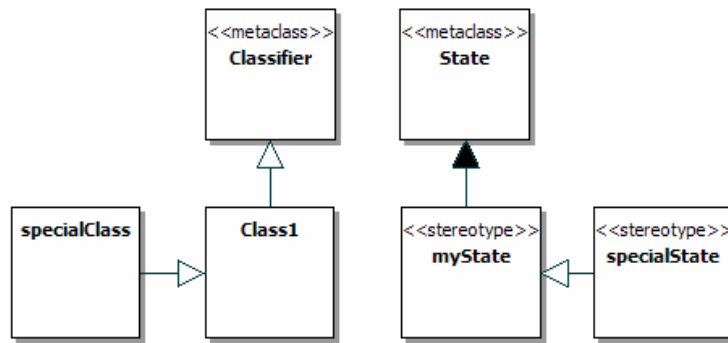


Figure 27: Generalization and Extension Notation in the UML

Figure 27 gives an overview of the specialization concept in the UML. Classes and UML metaclasses can be specialized while stereotypes can only extend metaclasses. Generalization depicts the relationship (inheritance) between the general and the specialized class. It is denoted by a white filled arrowhead. In contrast, an extension is depicted by a black filled arrowhead. A stereotype can specialize another stereotype, but it cannot extend it. Note that the metaclass and stereotype headings are keywords and not stereotypes (because the notation within guillemots is the same).

Stereotypes can also define constraints in order to indicate restrictions. It is possible to define pre- or post-conditions, invariants and more. Constraints defined by extensions must not contradict the restrictions of the base class. In UML, restrictions can be expressed in any language, such as programming languages or natural languages. In this thesis, the Object Constraint Language (OCL) (see Section 4.5) is used for the profile definition, as it is more precise than natural language and can be evaluated by modeling tools automatically.

4.4.2 Requirements for a UML Profile

The UML offers the extension mechanism profiles to enable tailoring its use for a specific domain by means of a specialization with concrete, specific semantics and notations. UML profiles provide language extensions that specialize the meta-model of the UML and render its semantics and notations more precisely. However, UML profiles restrict this extension to UML to specializations of already existing UML language concepts. In essence, the model developed with a specific application still remains a compatible and valid UML model. This allows such models to be described and to be used through various existing UML tools. To develop a UML profile, concepts of the language have to be aligned to the UML concepts. The specialties of the language concepts have to be modeled in stereotypes, expressed through constraints, tagged values as well as additional notations and semantics. The main challenge for each language concept of the UML profile is to identify the most specific UML concept that still generalizes the original language concept, extend it and possibly constrain.

Furthermore, the UML Superstructure imposes several requirements for a profile. In the following, a brief overview on the most relevant requirements is given. The complete list of requirements can be found in the UML Superstructure document. First, a profile is not allowed to contradict with the already defined semantics of the extended model elements in the UML metamodel. The specialized model elements introduced by a profile shall only introduce more well-defined rules and may give additional constraints. They shall also not contradict with the constraints given in the metamodel. Of course, a profile shall enable the model repository interchange by means of XML datagram interchange between various modeling tools. UML profiles should allow to be applied and retracted from a model without rendering the model ill-formed.

4.5 The Object Constraint Language

The *Object Constraint Language* (OCL) [OMG05d] is a formal specification language used to express constraints and navigation within UML models. The OCL is part of the UML standard since the release of UML version 1.1.

The OCL has the power of the lower order predicate calculus including a simple set theory. It is mainly used to formalize bounds and constraints on objects of a model. These constraints must be valid for the overall system model. Besides, the OCL allows to check some properties on objects that are described within the model in a programming language independent way.

OCL is required, because a class model itself is incomplete to be able to act as means of specifying requirements for the classes at runtime. In the past, the additional definition applied to UML diagrams was done in natural, informal language. However, this was prone to ambiguous interpretation and not executable as code. It failed to address the main intention of simplifying and clarifying the class model. In contrast, OCL provides a very simple and intuitive language that can be used to express precise restrictions (*constraints*) for class and objects at runtime. It also provides a clear means to navigate the model. Traditional formal languages require a clear understanding of the mathematical foundation which would render it cumbersome and rarely well understood. The OCL has been designed to bridge the gap between a formal language that is precise and an intuitive language that can easily be accessed. The OCL consists of expressions which express the validity of the overall system model. The expressions are evaluated instantaneously. This implies that the model must not change during the evaluation nor the evaluation itself must modify the model. Therefore, the OCL is free of *side-effects* with the exception of preconditions and postconditions on methods.

The OCL does not specify its implementation and is therefore independent of any concrete programming language. In fact, the OCL is not a programming language but a declarative language. It is not possible to modify the control flow of a program and it is not able to execute operations that modify the system state. OCL expressions are qualifiers and not assignments. Operations are only allowed for querying of properties and are also guaranteed to be without side-effects. The OCL is a strongly typed language and expressions are only well-formed if all types involved correspond to the typing rules. The semantics of the OCL is formally defined in Annex A of [OMG05d].

4.5.1 OCL Context

Each OCL expression is specified in a context of an instance of a type. To refer to this instance, the OCL offers the keyword *self*. If the context is an operation, the formal parameters in the expression can be referred by the context. Frequently used constraints are the invariant constraints. Constraints specified as being invariant impose conditions to an instance of a type which must always be satisfied. The OCL offers the keyword *inv* for invariant constraints. Expressions which define an invariant constraint evaluate to a Boolean value within their given context. The following is an example of an invariant constraint on an instance of Plane:

```
context Plane
inv: self.engines > 0

context A380:Plane
inv: A380.engines = 4
```

The example shows an explicit context by the keyword *self* and the definition of a context reference by *A380*. If the reference is unambiguous, the explicit context reference can be omitted. Besides the

invariant constraints, constraints can be specified which have to be valid before and after the invocation of an operation or activity. The following example is a method which turns on the engines with the power percentage and returns the fuel which has been burned by the ignition:

```
context Plane::thrust(power: Real) : Real
pre: self.fuel > 0
post: self.fuel = self.fuel@pre - result
```

The *pre* constraint qualifies the condition which must be satisfied before the operation is invoked. In the above case the fuel must not be empty prior to the engine's ignition. Analogously, a *post* constraint specifies the conditions which must be satisfied after the execution of the operation. Both types of constraint have a Boolean type. The postfix *@pre* allows the OCL to get access to the prior value of the variable *fuel* which is considered to have changed by the invocation of the operation.

4.5.2 OCL Types

All expressions of the OCL have a type. The type determines which values are legal within a range and the methods which are allowed. Types that are defined in a model are also part of the types of the OCL. Basic types like numbers and strings are part of the OCL predefined value types. Within an expression access to properties or methods of the model is possible. The access to properties of basic types should be done by a dot '.' operator. The access to collection type should be done by using the arrow operator '->'. These collection type operations can also be used to access object. The objects are then implicitly converted to a set of a single object of the same type.

An expression in OCL is type valid if all types involved correspond to each other. If this is not the case, the expression is ill-formed. Types only correspond to each other if they are of the same type or a type is a subtype of the other. The same applies to the types if they are within a collection. Explicit type conversion is also supported by means of the *oclAsType(oclType)* method.

The following Figure 28 gives an overview of the available data types in the OCL:

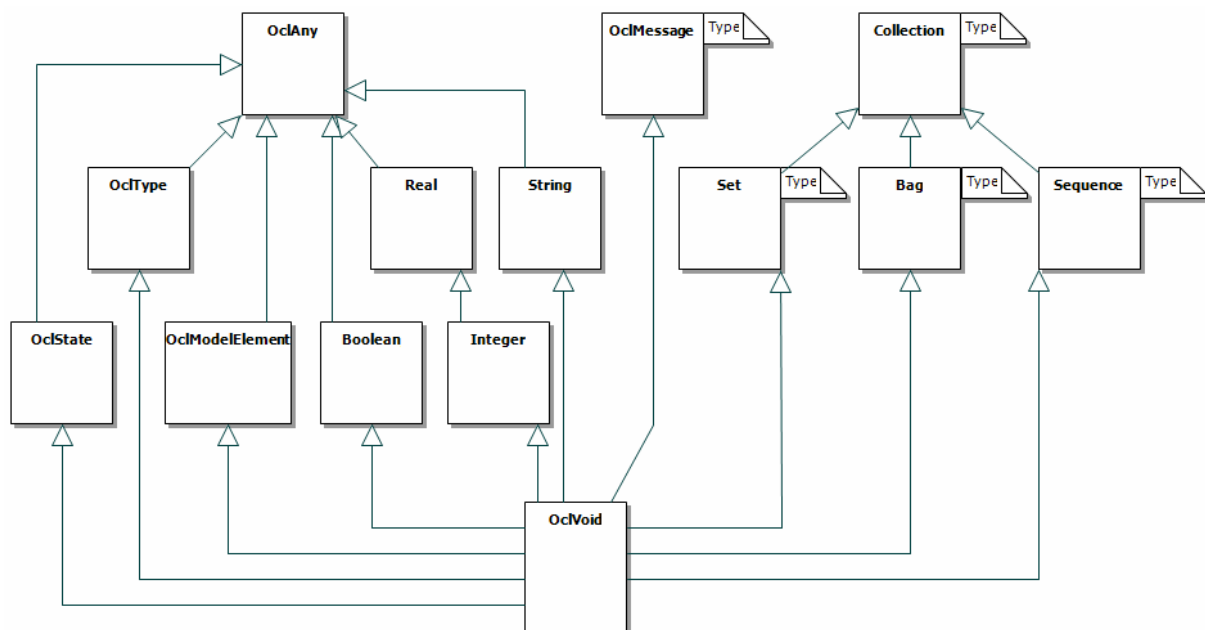


Figure 28: Data Types and Specialized Types in OCL

The types Collection, Set, Bag, Sequence and OclMessage are parameterizable data types. In essence, these data types compose several instances of a data type together.

4.6 XML Metadata Interchange

The *XML Metadata Interchange* (XMI) [OMG05e] is a standard for exchanging metadata information by means of the *Extensible Markup Language* (XML) [BPS+06] established by the OMG. Its application is to be used for the description of any metadata whose metamodel can be expressed in the *Meta-Object Facility* (MOF). As the UML is also based on the MOF, nowadays the most common use of XMI is to act as an interchange format for UML models. It can also be used for the serialization of models being an instance from other metamodels (languages).

The purpose of XMI is to enable easy interchange of metadata between UML modeling tools and to metadata based on the MOF repository. XMI is composed of the Extensible Markup Language (XML), the Unified Modeling Language (UML), the Meta-Object Facility (MOF) and a MOF mapping to XMI. The following Figure 29 shows the metamodeling layers of XMI compared to the metamodeling layers of UML.

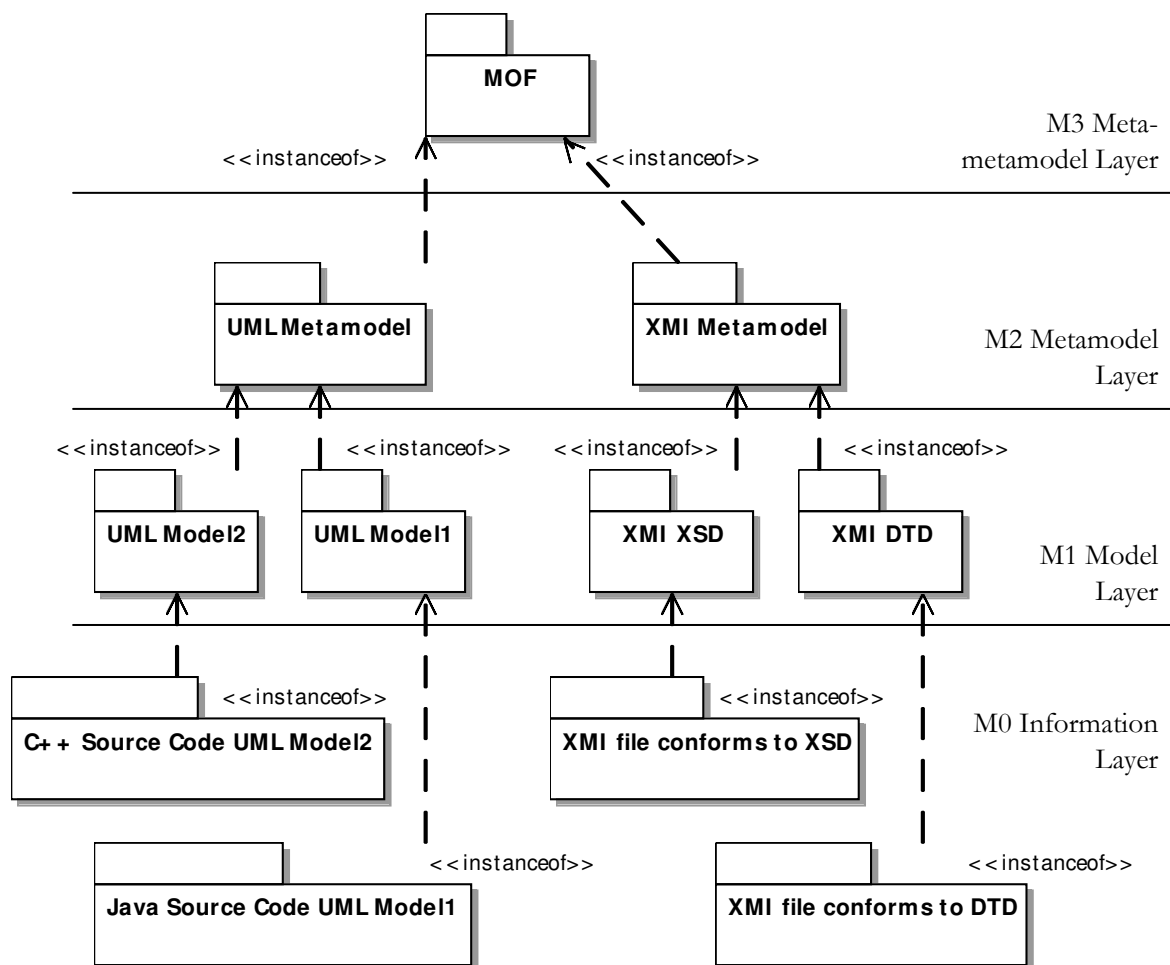


Figure 29: XMI Metamodel Layers

Several versions of XMI have been created whose main features are summarized in the following list:

- XMI 1.1 corresponds to MOF 1.3
- XMI 1.2 corresponds to MOF 1.4
- XMI 1.3 corresponds to MOF 1.4 with added schema support
- XMI 2.0 corresponds to MOF 1.4 with added schema support and changes the document format
- XMI 2.1 corresponds to MOF 2.0

4.7 Summary

In this chapter, the Unified Modeling Language 2 (UML 2) has been introduced. The profile described in this thesis provides an extension for tailoring this language to a specific domain. The first section has given an overview of the history and evolution of the UML. In the second section, a general overview of the UML 2 has been provided. The UML offers a total of thirteen different diagram types that can be categorized in structural and behavior types. The diagram types which are relevant for this profile were briefly described.

The UML is a meta-modeling language and is therefore a model for other models. However, the UML is defined by means of a meta-metamodel language, the Meta-Object Facility (MOF). This relation between these model languages has been described in the third section. The fourth section has explained the possibilities for extensions of the UML. The UML can be extended by means of metamodel extension or the profile mechanism. The requirements for a profile have also been noted in this section.

The UML uses the Object Constraint Language (OCL) to define constraints for the metaclasses. This specification language is also used for the profile in this thesis to define constraints and to define a mapping to SDL. Although the OCL is a stand-alone language, it is well aligned for its use within the UML. The OCL and its concepts have been described in the fifth section of this chapter. For the UML, a standardized diagram interchange format is defined, the XML Metadata Interchange (XMI). This XMI format is used for the prototype implementation of this profile and is the source for a mapping to an SDL specification. XMI and its features have been summarized in the sixth section.

5 Analysis of SDL with respect to Internet Communication Protocols

In this chapter, an analysis of SDL for its capabilities to specify and describe Internet communication and signaling protocols [WFH05, SWH06] is presented. The driving example is a multi-hop signaling protocol for Internet quality of service provisioning, namely the Resource Reservation Protocol (RSVP) [BZB+97] by the Internet Engineering Task Force (IETF). For clarity, only the main structural SDL diagrams are shown in this chapter. The complete SDL system specification of the RSVP model can be found in Appendix C: SDL Diagrams of the RSVP model.

With the era of the Internet and ubiquitous wireless access networks such as Wireless LANs [Rec04] and the Universal Mobile Telecommunications System (UMTS) [KAL+01] new network services have arisen. This circumstance also requires the development of new communication and signaling protocol concepts to cope with typical effects imposed by the mobility of nodes, data transmission losses and varying amount of communicating peers. The RSVP signaling protocol has been chosen for the analysis of SDL because of its soft-state concept, hop-by-hop signaling and local route repair mechanisms. As modern communication and signaling protocols for e.g. mobile ad-hoc networks (MANET) [MM04] and overlay networks [Dan04] use similar characteristics, RSVP can be conceived as technical precursor of this group of protocols.

5.1 Case Study of a Signaling Protocol: RSVP

For the last decade, a group of protocols has been designed using *soft state* for state maintenance. In contrast to hard state, a soft state itself expires if no periodical refreshes are received. Soft state protocols are expected to have less protocol complexity in state maintenance operations especially in extreme network situations. However, to best of the knowledge rare rigorous investigations have been performed on modeling these behaviors, especially for multi-hop soft state signaling protocols such as the Resource Reservation Protocol (RSVP). RSVP was the first soft state signaling protocol for Quality of Service (QoS) resource reservation developed by the Internet Engineering Task Force (IETF). RSVP specifications provide necessary message formats and processing rules for establishing and maintaining a state along a flow path. However, the same as most of the follow-up soft state signaling protocols, the RSVP specification does not describe in detail how a link failure is detected and circumvented.

The following presents a formal model of a soft state signaling protocol based on the Specification and Description Language (SDL). It investigates the RSVP protocol as a case study and particularly with respect to route changes. The model is built on a simplified IP layer model for RSVP message routing. Different from existing modeling approaches the model allows an easy change of the analyzed network scenario without the need of any re-specification of the SDL router blocks. There is no centralized entity responsible for routing, avoiding the necessity of re-specification for any new network topology. It is shown how the RSVP state recovery is verified and validated. This modeling approach is also useful for the validation, modeling and analysis of soft state protocols in general and for simplifying language constructs for this UML profile.

5.1.1 Studies on soft state protocols

System designers argue soft state is *better* than hard state and using soft state the handling of network condition changes is *easy* [SEF+97, RM99]. However, these claims have been more based on intuitive, high-level thoughts and explanations, instead of formal, exhaustive modeling and analysis. In contrast to the original expectations, soft state protocols being developed so far are still far from being simple, especially when coupled with channel reliability, multicast sessions or traffic control models. Soft state protocols developed so far can be categorized into two types: End-to-end protocols and hop-by-hop protocols.

The former only involves certain type of state in an end-to-end way, without bothering any other nodes in-between; examples of this type include the RealTime Control Protocol (RTCP) [SCF+03] and the Session Initiation Protocol (SIP) [RSC+02]. Hop-by-hop protocols, such as RSVP and Next Steps in Signaling (NSIS) framework [HKL+05], on the other hand, involve states in one or more router(s) in-between in addition to the states in the communicating ends. The latter is more representative and more comprehensive to demonstrate the soft state operations. So, this is chosen as the example for general discussions of soft state. Given the particular importance of soft state protocols, there have been recently a few efforts on their modeling and analysis. Raman and McCanne [RM99] presented a model for the soft state notion based on Jackson queuing networks; a performance study of hard state and soft state signaling protocols was performed by Ji et al. [JGK+03]. However, more detailed formal modeling and validation is still missing. A general formal soft state protocol analysis has been presented in [FH05], but a concrete analysis of an existing soft state protocol is missing as well. Therefore, it serves well the purpose of analyzing the capabilities of a protocol specification and description language.

5.1.2 Overview of RSVP

RSVP aims to provide end-to-end quality of service (QoS) signaling for application data streams. Hosts use RSVP to request a specific QoS from the network for particular application flows. Routers use RSVP to deliver QoS requests to all routers along the data path. RSVP can also maintain and refresh states for a requested QoS application flow. RSVP carries QoS signaling messages through the network, visiting each node along the data path. If the reservation succeeds, the RSVP module sets parameters in a packet classifier and packet scheduler to obtain the desired QoS. The design of RSVP distinguishes itself by a number of fundamental ways, particularly, soft state management, two-pass signaling message exchanges, receiver-based resource reservation and separation of QoS signaling from routing [ZDE+93].

Due to the fact that the flow of delivery paths might change during the life of an application flow, RSVP takes a soft state approach in its design, creating and removing the protocol states (*Path* and *Resv* states) in routers and hosts. RSVP sends periodic refresh messages (*Path* and *Resv*) to maintain its states and recover from occasional message loss. In the absence of refresh messages, the RSVP states automatically time out and are deleted. RSVP is not a routing protocol but rather is designed to interoperate with current and future unicast and multicast routing protocols. While routing protocols are responsible for choosing the routes to use to forward packets, RSVP consults local routing tables to obtain routes. It is only responsible for reservation setup along a data path.

5.1.3 Formal Process

Traditionally, IETF protocols, namely the Request for Comments (RFC), are specified in a textual, informal format. A formal description using SDL of such a protocol can help to specify the functional

operation clearly and unambiguously. It allows to detect protocol anomalies or design errors like deadlock or livelock situations more easily. Previous studies like [MSP01, CB03, CGM+04] presented analysis and validation of several IETF protocols using formal description techniques.

However, their analyses were limited to a single or only very few fixed use cases. They were only applied to protocols operating in an end-to-end fashion or using hard state in principle. None of them investigated any soft state signaling protocol, nor considered randomly chosen link failures. It is argued to be important to guarantee the proper protocol operations in dynamic environments, especially that soft state signaling protocols are error-free and also precisely presented for the correctness of implementations. In this thesis, a modeling approach is developed that proves that despite interactions between the possibly dynamic chain of intermediate hops and random link failures the correctness and robustness in soft state protocols can still be proven by way of formal description and validation. Besides of this, the efforts are shown which are required to work-around the shortcomings of SDL for this purpose.

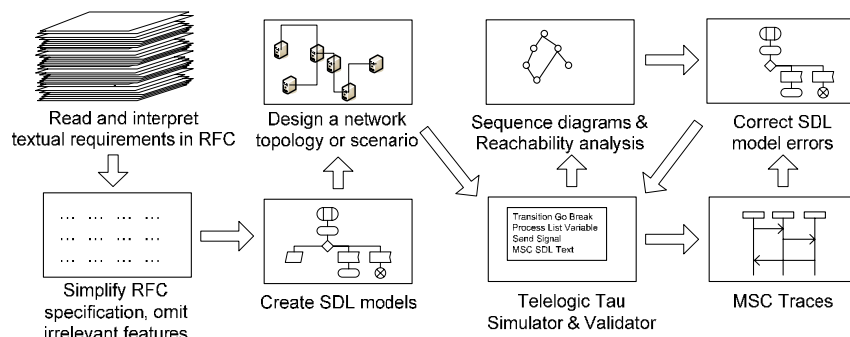


Figure 30: The Formal Process

Figure 30 pictorially represents the formal process that has been used for modeling – starting with reading and interpreting the RFC. After the interpretation of an RFC, the SDL models are specified and a network scenario is created.

An integrated SDL development tool, Telelogic Tau 4.6 [Tau], is used for the formal verification and validation of the created model. RSVP features that do not directly relate to route change detection and recovery were chosen not to be specified to reduce complexity (e.g., reservation filters). A network topology was created that is assumed sufficiently complex for showing the route re-establishment functionality of RSVP. Due to the decentralized IP network layer architecture additional network topologies can be created and analyzed without the need to re-specify the SDL models.

5.1.4 SDL Modeling of Message Routing in IP Networks

To the best of the authors knowledge, formal models developed so far focused on IP based networks either model end-to-end protocols which are formally specified for their special purpose; or simply three entities are assumed: A sender, a recipient and a general transport block as a centralized entity for routing, forwarding and packet loss modeling (e.g. [MSP01, CB03, CGM+04]). These approaches have the disadvantage that for each new network topology the central routing entity or intermediate nodes have to be re-specified and adapted to the new network configuration. Additionally, link failures are hard to emulate. Multi-hop protocols with route failures cannot be modeled using a centralized entity or fixed formalized nodes for message transport.

Therefore, it is proposed to specify a formal decentralized network layer architecture which automatically notices its neighboring entities and reachable destinations by itself. Modeling of IP

based communication protocols in dynamic network topologies suffers by some SDL shortcomings and limitations of Tau. SDL does not offer a dynamic amount of channels connected to a block. Therefore, the router models have a fixed amount of three channels (network links) available. Furthermore, SDL does not provide native support of IP addresses. Instead, SDL process IDs (*Pid*) are used for addressing of nodes in the topology. It is assumed that this is no downside of the model if small network topologies without the need for special routing are required. The signal *myPID* is used to announce the destination node's address (process IDs) to other nodes. In reality, this is defined by the user itself or by the user's application.

The routing algorithm is inspired by *Distance Vector Routing* protocols like the *Routing Information Protocol* (RIP) [Mal94]. To reduce complexity, the periodically broadcasted distance vector updates are replaced by signals which trigger distance vector updates between neighboring routers. This feature is especially useful for not being confused by minor relevant network layer messages if the upper layer's soft state protocol messages are to be analyzed and validated. Furthermore, this is required for formal analysis using the Tau Validator. The Validator does not include signals from the environment if any transitions are still scheduled. While the routing tables are updating, the system converges to stable state. Some more enhancements and simplification have been undertaken to bypass known Distance Vector Routing problems (like the *count-to-infinity* problem [Tan02]). They are not discussed here in detail as this is out of the scope of this thesis. Note that the model of RSVP is intentionally not bound to any specific IP routing protocol, so the use of a modified routing protocol here does not violate any RSVP requirement.

The IP routing layer is modeled as a block consisting of a *forwarding* and a *routing* block. The basic operational principle is the following: The *forwarder* receives *datagrams* which is an SDL structure consisting of the variables *Source* (sender of this packet), *Destination* (destination for this packet), *Phop* (previous hop) and a payload *msg* from the upper layer – RSVP messages in this case. If the *forwarder* receives a datagram, it queries the *routing* block for the address of the next hop and forwards the *datagram* to this hop. Routing table updates are received by a special signal *DistanceVector* containing the routing table of the neighbor's routing layer. This information is used by the *routing* block to update its local routing table.

The investigated scenario consists of one *NI* (network initiator), multiple *NFs* (network forwarder with routing functionality) and one *NR* (network receiver). The *NI* is the entity that generates RSVP messages and tries to establish a reservation state along the path from *NI* via multiple *NFs* down to the receiving *NR*. Every single hop on the path establishes a requested RSVP state. All *NF* nodes have three connectors available for creating a network scenario. Unconnected signal channels have to be connected to *dump* blocks which silently consume all signals they receive. Figure 31 shows the SDL system with the described network topology. The message flow used in this scenario is from *NI* down to *NR* via several *NFs* and vice versa. Note that the shortest route between *NI* and *NR* is via *NF1*, *NF5* and *NF3*. After a possible shutdown of *NF5* an alternative route is established via *NF1*, *NF2*, *NF4* and *NF3*.

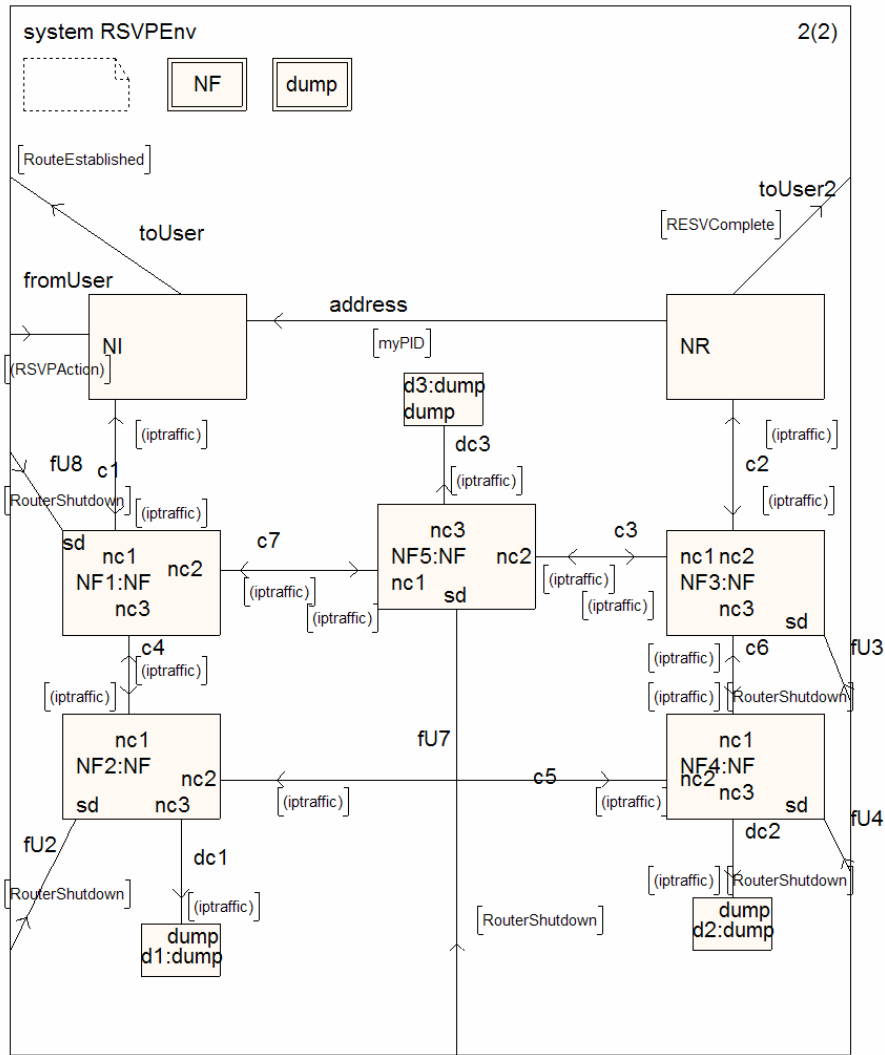


Figure 31: RSVP Network Scenario Model generated in Tau 4.6

All routing layers feature an external *Shutdown* signal from the environment which allows the user to shutdown freely any or multiple instances. If a routing layer is triggered by such a *Shutdown* signal, it announces to shutdown by sending a *LinkFailure* signal to all its neighbors. Note that this is one modification to Distance Vector routing protocols which detect a node failure by the absence of the failed node's routing table updates. As periodic routing table update message add avoidable communication complexity to the scenario, the *LinkFailure* signal is introduced.

All neighboring hops are now trying to update their routing tables with new routing information and request table updates from their neighbors as well. The routing layer, once being shut down, is no longer operational and simply consumes each signal or message silently which it receives. The whole entity cannot operate anymore. This allows the analysis whether the soft state timing is able to maintain its state even if refresh messages are lost at the non-operational hop until the new route is established. See Figure 32 for an overview of the IP routing and RSVP block.

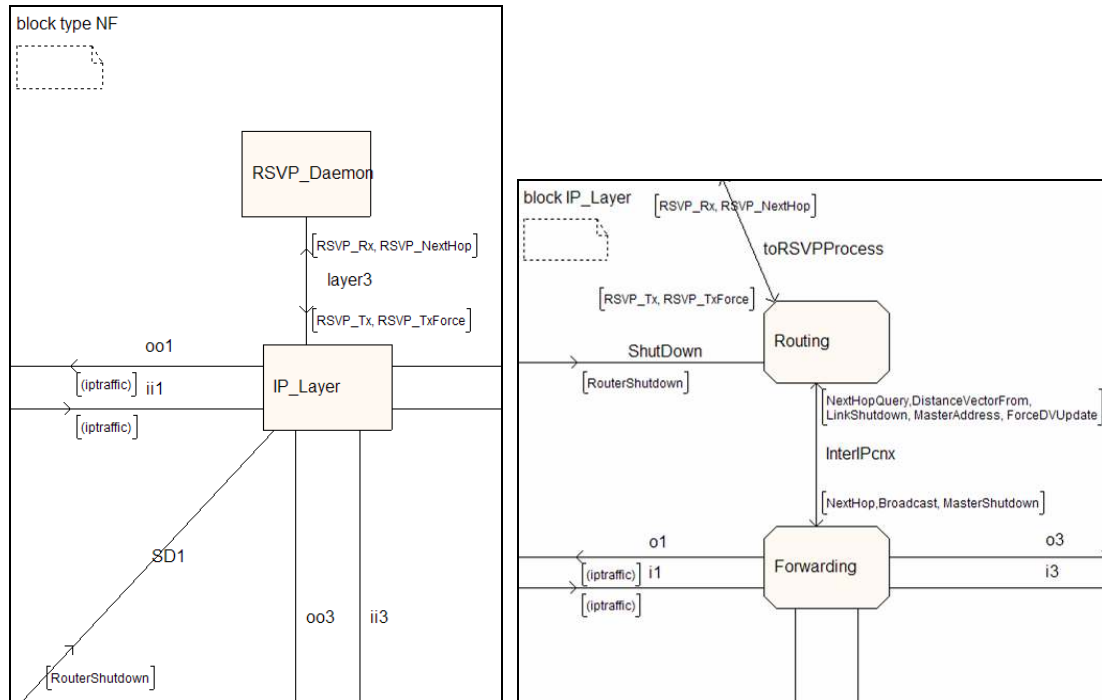


Figure 32: Internal Network Structure Block Type of all NF nodes

The NF block consists of a layered block structure which is the IP layer with routing functionality and a higher layer which is the RSVP daemon here. The *forwarding* block is responsible for receiving and forwarding datagrams. The *routing* block selects the next hop for a received datagram and maintains its local routing table. RSVP messages are sent to the RSVP daemon block by the *routing* block when being received.

5.1.5 Formal Analysis of RSVP State Maintenance with Link Failure

The previous section has described the SDL models of RSVP that can restore a valid path after a link failure with Tau. When the simulation is started, the system announces that it is ready for operation by sending a special signal and all routing tables are build up to allow a complete routing between all nodes. The *NI* accepts three different signal triggers from the environment: *RSVPStart*, *RSVPTeardown* and *RSVPStop*. *RSVPStart* begins creating a path state and resource reservation along the path down to the *NR*. The *NI* periodically sends new path messages to keep the RSVP soft state alive. The *RSVPTeardown* signal triggers the *NI* to stop sending refresh messages and to send a *PathTeardown* message towards the *NR*.

All nodes in-between delete the associated states from this reservation and forward the teardown message to the next hop (*Explicit Teardown*). The *RSVPStop* signal just stops the *NI* from sending new state refresh messages towards the *NR*. This leads to a state timeout at all hops and the states are deleted after the state lifetime expiry.

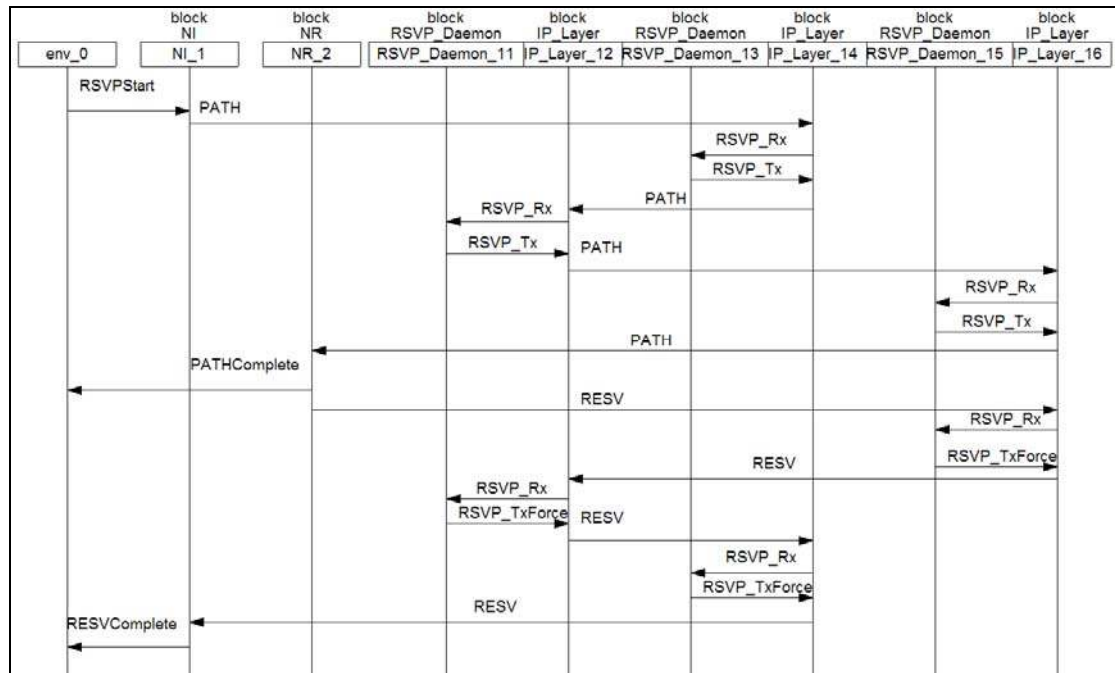


Figure 33: Message Flow of a RSVP Resource Reservation

In Figure 33, a default RSVP *Path* and *Resv* message refresh flow is shown. The *Path* message is sent downstream from the *NI* hop-by-hop to the *NR*. The corresponding *Resv* message is sent upstream from the *NR* hop-by-hop to the *NI*. Message parameters and process states are not shown for clarity. In this excerpt of a MSC, the message exchange is shown from the *NI* via *NF1*, *NF5* and *NF3* down to the *NR*. This is the shortest path. Notice the internal message exchange between the IP Layer and the RSVP instances. The RSVP daemon is notified of the reception of RSVP messages by the *RSVP_Rx* signal and itself sends a RSVP message using the *RSVP_Tx* or *RSVP_TxForce* signal. While the *RSVP_Tx* signal allows the IP layer to select the next hop, the *RSVP_TxForce* explicitly addresses the next hop. The Tau SDL simulator is used to trace the correct establishment of *RSVP_Established* states in all RSVP intermediate hops and the *NR*.

Next, a router shutdown is triggered. In this scenario, *NF5* is selected as the failure hop. By doing this, an alternative route has to be established by the IP routing layer. After the *NF5* has been shut down, it announces its decease by sending a *LinkFailure* signal to all its neighbors. Because of this signal, they try to update their local routing tables with their neighbors as well.

The following Figure 34 presents an excerpt of a MSC that shows the message exchange in case of the *NF5* shutdown. After the router is shut down by a *RouterShutdown* signal (top left in Figure 34) all neighboring routers try to update their local routing tables. Note that the *Resv* message already on the way up to the *NI*, is discarded by the link failure and is lost (marked by a dotted arrow). While the routers try to update their information, a newly created RSVP message is lost while being routed, visible at the signal marked with the dotted arrow.

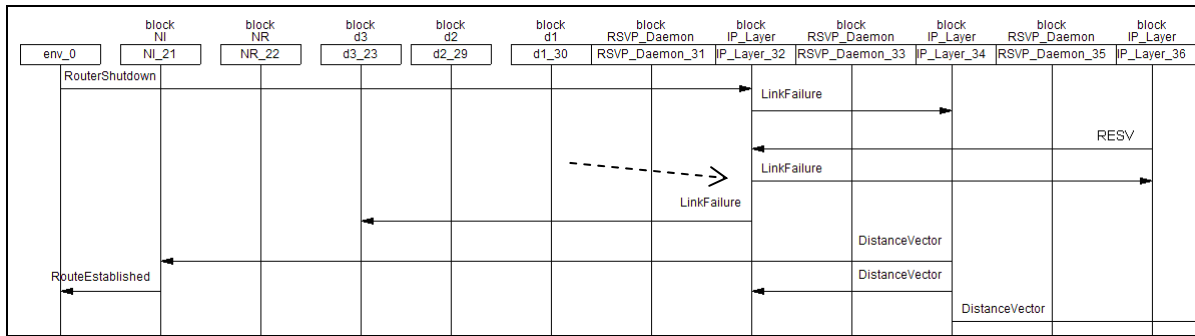


Figure 34: Initial Message Exchange after Router Shutdown

Most of the messages shown in Figure 34 are routing table updates used for the establishment of the new path between *NI* and *NR*. They are not discussed in detail here.

The next RSVP refresh message is due shortly after the new route has been established and is shown in the following Figure 35. This excerpt of the MSC is the time continuation of the signal exchange shown in Figure 34. Some routing table update messages and inactive instances are skipped. The RSVP message (*Path*) is delivered via the alternative route *NF1*, *NF2*, *NF4*, *NF3* down to the *NR*. This is accomplished using normal path state recovery initiated by the next refresh message from the *NI*. One can see that the message is correctly routed through the new hops of the alternative route. The SDL simulation confirms that all new hops are able to establish a correct *RSVP_Established* state. The RSVP soft state operation continues with correct behavior. This is caused by the detection of the route change if the previous hop of the new RSVP message differs from the one which has been recorded on previous RSVP messages. The same detection applies on changes of the next hop which is decided by the IP routing layer. This operation has been validated using the built-in Validator of Tau using exhaustive state space exploration.

Note that the RSVP model does not include RSVP features like multicast and the admission and policy modules since these are not particularly interesting for route re-establishment. RSVP multicast adds a high level of complexity to the protocol design and multicast support (actually one of its succeeding IETF efforts, NSIS, has decided to remove multicast from basic signaling support), thus it is not considered here. Therefore, multicast related operations like merging and styles processing were not considered. *Local repair* has been implemented which improves route recovery by immediately sending *Path* and *Resv* messages towards the previous and next hop if a route change is detected.

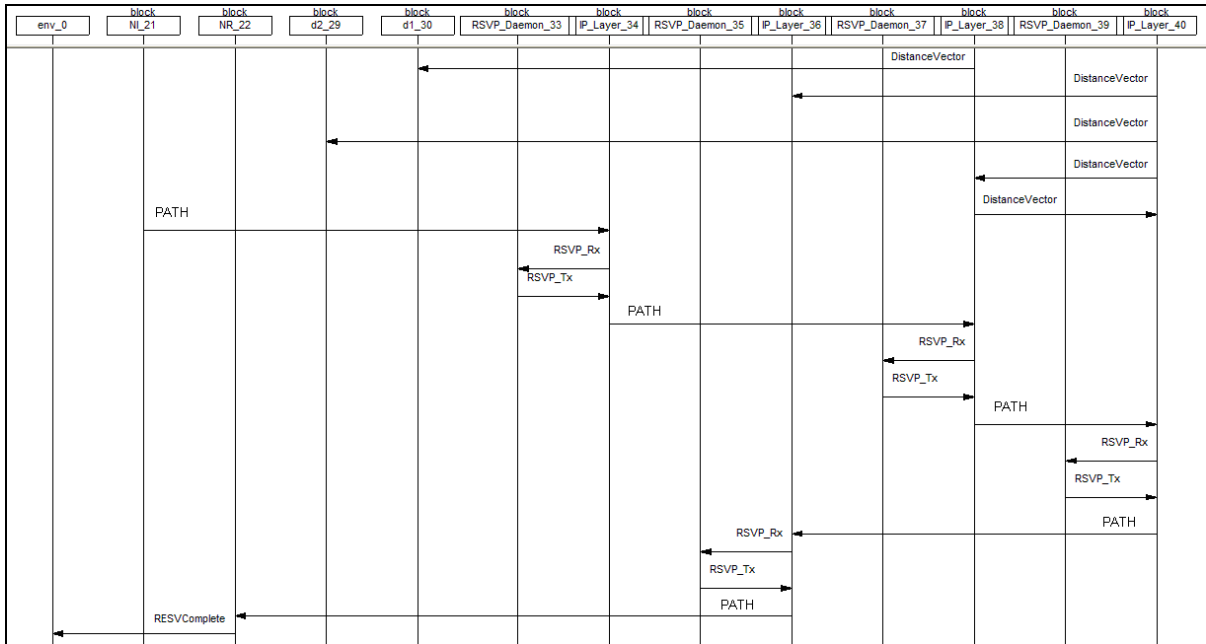


Figure 35: New Route and State Establishment

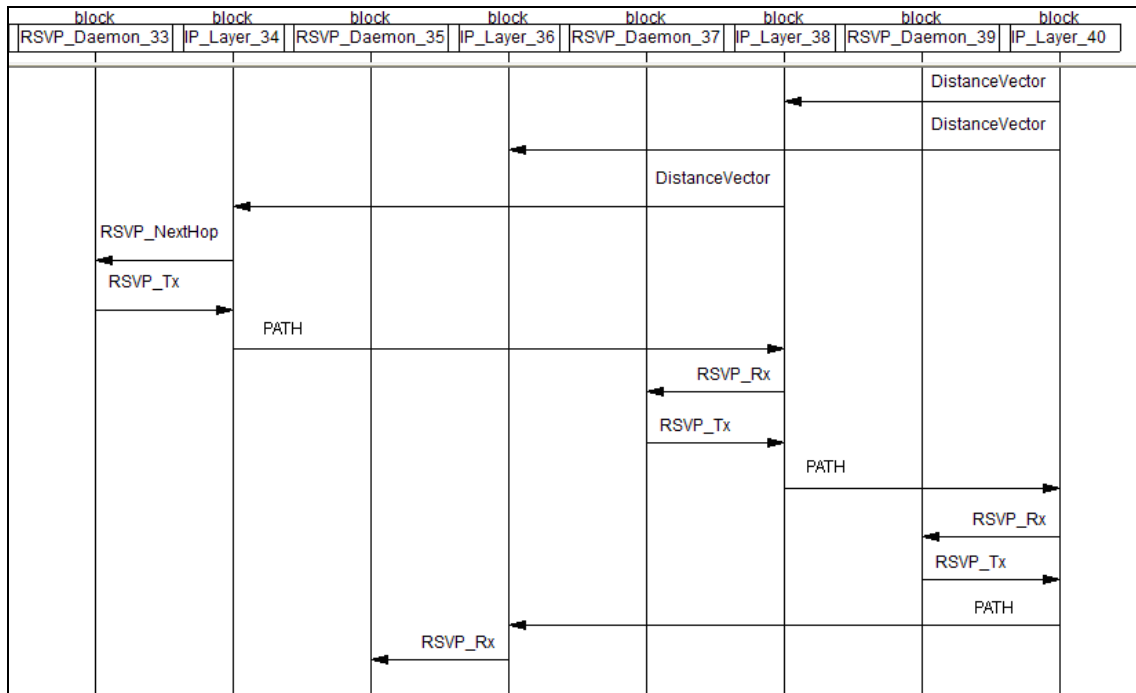


Figure 36: Excerpt of the new Route Establishment Message Flow using Local Repair

Figure 36 shows an excerpt of a MSC with a local repair action triggered by routing table updates. This excerpt of a MSC is an alternative time continuation of the signal exchange shown in Figure 34. Some routing table update messages and inactive instances are skipped. The new RSVP message (*Path*) is delivered via the alternative route *NF1*, *NF2*, *NF4*, *NF3* down to the *NR*. Note that a *Path* message is sent triggered by the routing table update by receiving the *DistanceVector* signal. The RSVP process is notified of the route change by the *RSVP_NextHop* signal.

5.1.6 Conclusions

The simplified SDL model of the RSVP multi-hop signaling protocols has shown some of the limitations of SDL that renders it cumbersome for this specific purpose. As shown in the SDL structural model of the *NF* and *NR* with *NI* hops, dummy instances had to be created to allow description of a network topology generated from a generic block. In addition to the considerable efforts, this makes simulation and analysis more complicated.

RSVP is a soft-state protocol. This feature has been implemented by timers which are started when a soft state being entered and a signal input that waits for the timeout. This is a low-level work-around and does not abstract from the main intention of a soft state. For protocol engineering purposes a timer is merely required for proper termination in case of errors or timeout. This timer has a different intention than recovery from a previous failure. Therefore, a more abstract high-level view on this concept is required which focuses on the intended behavior of this timer. Besides of the soft state itself, also soft-variables are a useful feature which can be found in a few soft state based signaling protocols. However, this is not implemented in this profile as this is nowadays a rather rarely used concept. In addition, it is believed that it may render simulation and analysis in a more complicated way instead of being simple.

To study the robustness of a communication and signaling protocol in a packet switched network it is also of great interest if a loss of packets leads to erroneous behavior and how the protocol can recover from routing effects, such as path changes. While the first issue is not covered in the model the latter issue can be triggered by hand. Of course, this may lead to not equally distributed event occurrences of the path change. This could lead to the conclusion that the protocol behaves correctly but some concrete timing has not been covered by simulation. As SDL does not explicitly supports randomness, this can only be done manually or non-deterministically.

The following section gives insights into the analysis and rationale of the features that are consequently added to the UML CS profile.

5.2 Language Concept for the UML CS Profile

SDL is the first language for specification, design and development of real time systems and in particular for telecommunication applications. Nevertheless as evaluated in the previous section, it is cumbersome for the current and upcoming communication protocol engineering for packet switched networks. Current packet switched networks like the Internet or mobile wireless access networks (such as the Universal Mobile Telecommunication System and beyond) demand for new methodologies when modeling protocols. This includes such typical effects like communication path route change, robustness to message losses, roaming of computer with handover as well as the specification of communication protocols for multi-hop overlay networks and multi-hop signaling.

SDL-2000 is the first target language for UML CS profile's model elements. All UML CS model elements in the class, composite structure, activity or state machines, are mapped to a corresponding SDL element. However, the design principle of the UML CS profile is

- to enable mapping from UML models to formal specification and description languages – not necessarily limited to SDL.
- to focus the language on a small, simple set with only a few language elements to learn.

Therefore, some advanced SDL features of SDL-2000 are currently not supported, as this *may* not be equivalently mappable to other formal description techniques. Partially, this is inspired by the previous

publications in [Art01, Gra03, KLP+04, KPK+03, She05] where some features of SDL are proposed to be removed and some others are requested to be added. In summary, the following language features of SDL-2000 are currently not supported by the profile – however, this may be done in future versions of this profile:

- Exceptions,
- Templates (context parameters),
- State aggregation,
- State types and
- Optional transitions,
- Redefinition of state machines and channels.

Exceptions are a very handy language construct which gives support for dealing with unusual or unexpected conditions. Nevertheless, exceptions are not part of previous versions of SDL and other FDTs. In addition, the use of exceptions can be avoided by careful system design. They can also be emulated by use of signals although with a bit more effort. Therefore, exceptions are currently not supported to enable a feasible mapping.

Context parameters are similar to class templates in object-orientated programming. This allows specifying agent types as formal parameters of agents. Due to the complexity this does not seem to be very popular in available SDL-2000 system descriptions.

State types and aggregations allow typing and decomposition of states. State types enable redefinition and virtualization of states. This is a consequent result of the new agent concept of SDL-2000. State aggregation is a particular form of composite state. The state aggregation construct is the replacement of the service construct of SDL-92.

Furthermore, optional transitions are currently not supported. An optional transition represents a static conditional branch in the behavior description. The condition of the branch is evaluated before any execution of the system. The UML CS profile only supports dynamic conditional branches at this point. This can be used as a replacement for static branches. Redefinition of a specialized state machine is not supported, although this enables to modify the behavior during specialization. However, this profile semantics expects that specialized agents react identical to their parent agent when triggered by the same sequence of events. This is especially important if an agent type is replaced by its sub-type and still has to show the identical behavior. Channel refinement (redefinition) is also not supported.

Recently, the formal specification of IP network based communication protocols has gained more attraction. Traditionally, IETF protocols, namely the *Request for Comments* (RFC), are specified in a textual, informal format. A formal description using a formal language like the UML CS profile or SDL of such a protocol can help to specify the functional operation clearly and unambiguously. It allows detecting protocol anomalies or design errors like deadlock or livelock situations more easily. Previous studies like [MSP01, CB03, CGM+04] presented analysis and validation of several IETF protocols using formal description techniques. The fact that formal methods can help to detect protocol design errors has been shown in [BBK02] for example. In this work, possible deadlock configurations have been discovered in the interoperation of different http server versions.

Current developments towards all-IP networks underline the expectations that IP network communication protocol modeling will gain much more attention in the next years. Therefore, it is necessary to add new features to the language. They are aligned to ease the development of IP based networks and communication protocol models. With the recent experience gained in modeling multi-

hop Internet signaling protocols, some shortcomings in SDL have been identified which render some features of the Internet hard to formalize. This especially applies to robustness testing of communication protocols and to the specification and validation of multi-hop signaling protocols. However, in [SWH06] it has been presented that mobility and roaming issues in IP-based networks can be well described by means of SDL.

The following features have been identified to be necessary or useful to be added to UML CS profile allowing exhaustive IP communication protocol specification and analysis:

- definable randomness with a basic set of distribution functions,
- consumption of input signals only from a distinct address or connection source,
- creation of dynamic connections for various network topologies and
- native soft state support.

In the following, the rationale is given and discussed why these features are added to the UML CS profile which are not an integral part of the SDL standard specification.

5.2.1 Randomness

For robustness analysis of communication protocols, it is important to examine a communication protocol's ability to deal with packet losses. It is cumbersome to model packet loss probability in the Internet by means of the non-deterministic features of SDL, but can be modeled approximately using specific distribution functions. Unfortunately, SDL does not offer any random functionality at all – except for the *none* spontaneous transition and the *any* nondeterministic decision. Therefore, some SDL tools offer a proprietary support for randomly generated values by including several libraries to SDL.

Non-determinism is much too rough for packet loss modeling in the Internet. So a random function is introduced to UML CS that allows specification of a more precise randomness. The concept is mainly taken from the *Tau* [Tau] tool's random library, because it has shown to be useful and applicable.

First, an abstract data type called *RandomControl* is defined which allows generating pseudo-random numbers. Each *RandomControl* variable has to be initialized by the *DefineSeed(Integer)* function. *Integer* should be an odd value in the range 1 to 32767. When using the same integer value the same random number sequence is generated. That implies that sequences of random numbers are reproducible while the initial seed value remains unchanged.

A number of distributions are supported by following functions:

- *Random(RandomControl)*

is the basic random generator.

- *Erlang(Mean, N, RandomControl)*

provides Erlang-N distributed random numbers.

- *NegExp(Mean, RandomControl)*

provides negative exponential distributed random numbers.

- *HyperExp(Mean1, Mean2, Alpha, RandomControl)*

With probability *Alpha* it returns a negative exponential distributed random number with mean *Mean1* and with the probability *1-Alpha* it returns a negative exponential distributed random number with mean *Mean2*.

- *Uniform(Low, High, RandomControl)*

returns uniformly distributed random numbers in the range *Low* to *High*.

- *Geometric(p, RandomControl)*

returns geometric distributed random numbers with the mean $p/(1-p)$.

- *Draw(Alpha, RandomControl)*

returns true with the probability *Alpha* and false with the probability $1-Alpha$.

- *RandInt(Low, High, RandomControl)*

returns one of the values *Low*, *Low+1*, ..., *High-1*, *High* with equal probability.

5.2.2 Input From/Via

While SDL provides explicit addressing of signals for output using the following textual expression

```
'OUTPUT' <signal name> [<actual parameters>] ['to' <address>] ['via'
<signal route name>]
```

there is no corresponding construct for the reception of a signal available in SDL. The reception of a signal *via* a specific signal route name or channel name cannot be determined. The reception of a signal from a specific process is possible if the SDL state machine variable *sender* is evaluated. This variable is updated when signal has been consumed with the process identification of the process that has sent the signal.

Therefore, an extended *INPUT* expression is introduced to this profile with the optional attributes *FROM* and *VIA*. Using this new *INPUT* construct, a signal is only consumed if the process identification address matches the sender's or the signal is received on a specific gate (port). Notice that the sender's address or signal route is evaluated *before* the signal is consumed while evaluating the *sender* variable is only possible *after* consumption of the signal.

This *INPUT FROM/VIA* construct is helpful if only gates (network interfaces) are relevant for message passing or addressing. One might think of a network bridge where it is only necessary to know on which side the message has been received.

5.2.3 Dynamic Gates/Ports

Recent studies in the modeling and robustness analysis of multi-hop Internet signaling protocols have shown that SDL is not well suited to create certain network topologies. IP network topologies require the free placement and interconnection of router nodes in-between the signaling path. Multiple routes from the network initiator downstream to the network recipient are necessary for the study of the robustness of signaling protocols. Such a model has already been developed in [WFH05] where considerable efforts have been undertaken to circumvent the shortcomings of SDL.

To create multiple hop network topologies, two modeling approaches are imaginable. It is assumed that multiple intermediate network nodes have to be modeled by a block type. A single gate is defined and all communications between all neighboring nodes are directed through this gate. Alternatively, multiple gates are defined for each single communication between the current node and one distinct neighboring node. Following Figure 37 and Figure 38 depict examples of both modeling approaches.

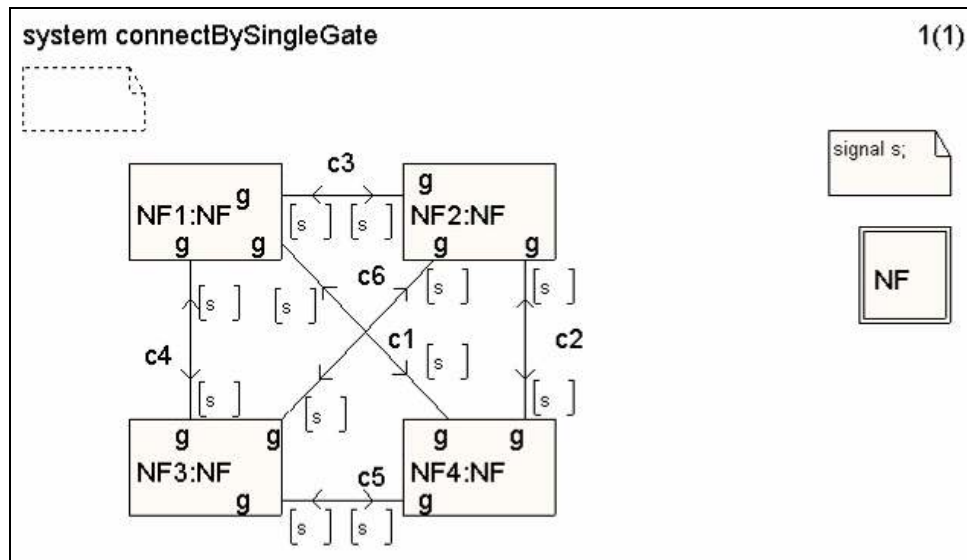


Figure 37: Block Interconnection in a Multiple Node Network using a single Gate

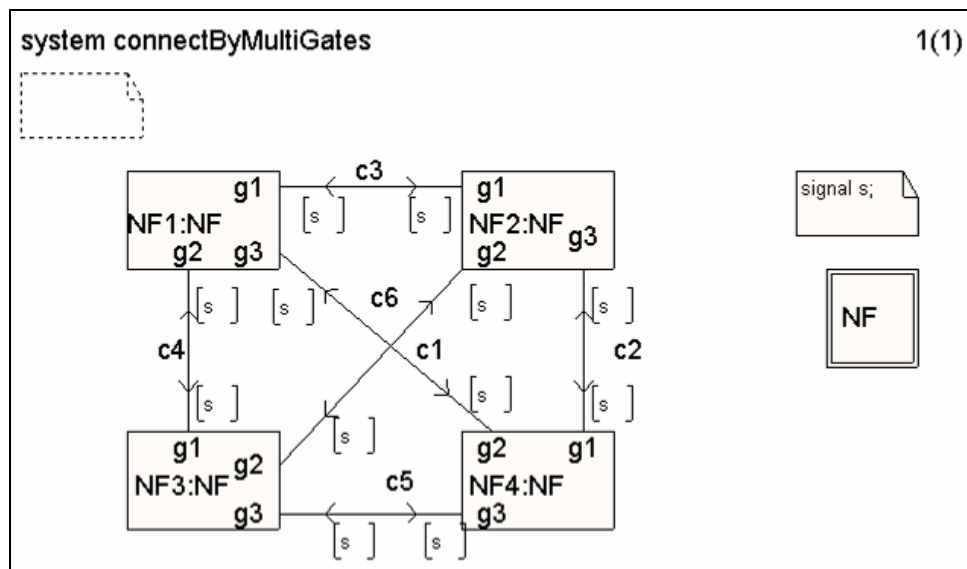


Figure 38: Block Interconnection in a Multiple Node Network using multiple Gates

The process model of the block type *NF* is not shown here as it is not relevant. Note that both SDL systems are syntactically and semantically valid SDL models and are both executable for validation purposes. Both systems implement the following:

The process model and the interconnection gates of the blocks are specified in the block type *NF*. All four blocks, derived from *NF*, are connected to each other for a fully meshed connection. For simplification, only the signal *s* is valid on the channels. In Figure 37, the channel connection is made via a single gate *g* in all blocks. In Figure 38, the channel connection is made via the three distinct gates *g1*, *g2* and *g3* for each channel connection. The main advantage of the single gate version is that new blocks can simply be added to the topology. The channel for the connection is set between the two gates *g* on both blocks.

In the system in Figure 38 this is not possible, because a new block has no spare gate at any of the already present blocks. Thus, the block type *NF* has to be re-specified adding a new gate *g4*. This

introduces another problem since all present blocks also inherit the new gate *g4*. SDL does not allow unconnected gates. Therefore, new blocks have to be defined which are connected to the new gate *g4* on all blocks. In [WFH05] this has been achieved by the introduction of *dump* blocks which silently consume all signals they receive.

Unfortunately, the single gate approach has a major downside which renders communication between the blocks impossible. The communication between the blocks can only be addressed using the *OUTPUT s TO <process id>* construct. The *OUTPUT s* or the *OUTPUT s VIA g* instructions do not specify the destination and there are multiple ways possible. The SDL standard specifies nondeterministic delivery of the signal to a reachable process in this case. Therefore, the recipient is only known after a process has received the signal. The *OUTPUT s TO <process id>* instruction implies that the *process identification (Pid)* of the receiver is known before the instruction can be executed. The Pid of an SDL process can only be determined if a signal from this specific process is received and the variable *sender* is evaluated. Hence, no process within a block can acquire the Pid from any other connected block's process. A solution to this problem would be a signal broadcast but such an explicit broadcast addressing is currently not supported in SDL.

```
OUTPUT s VIA ALL g1, g2;
```

Note that the above-mentioned *OUTPUT VIA ALL* is not a broadcast instruction. This instruction is shorthand for the following SDL instruction (except that a repeated signal evaluation of *s* is skipped):

```
OUTPUT s VIA g1;
OUTPUT s VIA g2;
```

The *OUTPUT VIA ALL* instruction is even not supported in SDL-2000 anymore. Consequently, the solution of acquiring the neighbor block's process identification is possible with the usage of multiple gates. As already mentioned this introduces little amount of flexibility since additional block nodes require the specification of additional gates within the block type. Additionally, all currently defined blocks have to be connected to *dump* blocks with their newly added gates. *Dynamic gates* are introduced to allow easy specification of Internet network topologies. They add dynamic channel connectivity to blocks specified by block types. New gates can be added without the necessity for re-specification of the block type. The *dump* blocks are not required because dynamic gates do not require channel connections.

In the UML, the corresponding entity for a gate is a port. The detailed addressing of a dynamic port is defined in later sections. Note that the concrete addressed port is unknown. However, after signals have been sent, a distinct mapping between the process identification of the connected processes and the dynamic ports is possible.

5.2.4 Soft States

A number of protocols have been designed using soft states for state maintenance. In contrast to hard state, a soft state itself expires if no periodical refreshes are received. It is argued, soft state protocols have less protocol complexity in state maintenance operations especially with extreme network situations. Examples of soft state based Internet Protocols are the Resource Reservation Protocol (RSVP) [ZDE+93], Next Steps in Signaling (NSIS) suite [HKL+05], Session Initiation Protocol (SIP) [RSC+02] and more. Because researchers argue that soft state protocols are a highly attractive concept for Internet communication and signaling protocols, it has been decided to add soft state management concepts to this profile.

The basic concept of a soft state is the following: When a soft state is entered, the timeout value is evaluated and an internal timer is started. If the state is re-entered, the timer is re-started. If a transition is triggered by an event occurrence, the internal timer is stopped. If a soft state timeout occurs before the timer has been restarted, a transition is triggered.

Obviously, soft states can be alternatively created using timers and timer-triggered signals in standard SDL. Nevertheless, a more native and intuitive integration with a sufficient high-level view which abstracts from unnecessary details will increase the acceptance of this profile for Internet communication protocol modeling. This allows a direct mapping from UML CS to SDL using equivalent constructs with states and timers. It must be noted that in SDL-2000, a soft state can be specified by means of a state aggregation type with a specific value for an internal timer definition. This is a handy way to create soft states by means of the SDL language. However, this becomes cumbersome when there are multiple concurrent timers required for a specific state. This would require multiple soft state type definitions or composite data types for the state's parameters. In fact, a UML CS soft state should be mapped into these state types for SDL-2000.

Additionally, UML features a *SimpleTime* concept which enables to trigger a transition in a state machine after a certain amount of time has passed or a specific point of time has been reached. This concept is re-used in UML CS to feature the soft states mechanism.

5.3 Summary

In this chapter an Internet signaling protocol, namely the Resource Reservation Protocol (RSVP), has been specified by using the Specification and Description Language (SDL). This specification has been done with respect to the behavior of RSVP in case of message path route change and router shutdown. The re-establishment of a signaling path conformant to the specification was validated by simulation runs and visualized by means of Message Sequence Charts (MSC). This specification of a concrete Internet signaling protocol has been done to analyze the SDL language concepts that may be missing to allow an exhaustive model specification within a possible Internet network configuration. Due to the nature of the Internet, some missing language features were identified which render the specification process cumbersome by means of SDL. The following language constructs were found to be missing: Randomness, process or gate specific signal consumption, dynamic gates and soft states. However, some of these language constructs can be specified by SDL-96 and with SDL-2000. However, as noted this can be achieved only by very huge specifications that make simulation and analysis very difficult, complex and time-consuming as shown in the previous analysis of the RSVP specification.

An additional analysis of SDL has been done with respect to Internet mobility and roaming of mobile nodes within several Internet based network topologies in [SWH06]. However, this did not result in the identification of further missing high-level language features. Therefore, it has not been described in this chapter.

6 Overview of the UML CS Profile

In this chapter an informal introduction to the UML CS profile is given. It explains how a system can be described and which language constructs are available. The first section focuses on how to realize the description of a system architecture. The second focuses on the description of the behavior of the described entities; the third section gives an overview of the use and definition of data types. The following sections only provide a brief overview on how a system can be described by using a UML modeling tool and this profile. Hence, it is not a complete tutorial nor does it cover all modeling and language elements. The detailed syntax and semantics of the modeling elements is covered in the subsequent Chapter 7.

The UML is an object-oriented modeling technique and is consequentially type-based. All definitions within the model repository represent types. The actual instantiation of the system model takes place during the execution of the model. The system specification itself can be done by means of a modeling tool. UML diagrams are a handy way to visualize the model with a focus on several aspects. However, which properties are visible and accessible from specific UML diagrams is mainly dependent on the specific modeling tool that is used. Several UML modeling tools only support a *draw&print* paradigm, do not create a model repository and do not allow specifying the required properties for system modeling. These kinds of UML modeling tools are not suitable for the modeling process. The following Figure 39 illustrates the workflow sketch of UML CS-based modeling.

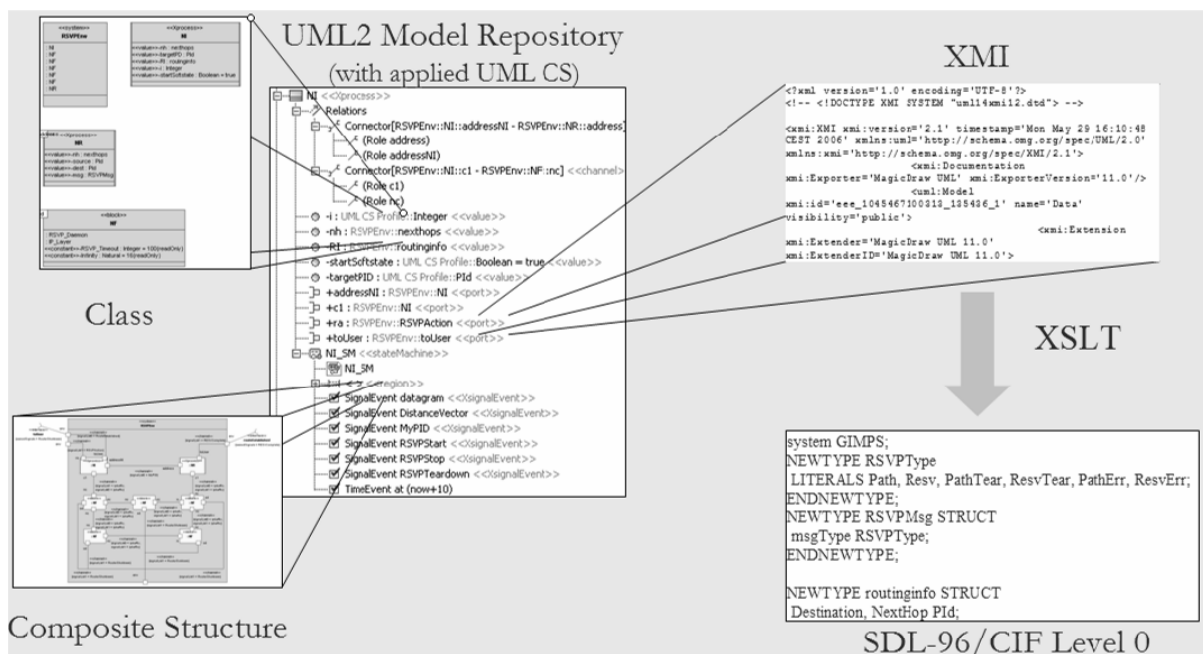


Figure 39: Modeling of a UML CS System

A UML model may have the UML CS profile applied. This allows defining specific extensions to the UML metamodel, although it remains a UML model. The model itself is represented by means of the UML model repository. This repository reflects the model defined by means of the UML metamodel. In particular, no diagram has to be created for defining a system, because a diagram only instantiates the abstract syntax of the model. The concrete syntax is visualized by means of diagram types.

In this Figure 39, two types of diagrams are shown which focus on specific aspects of the model. For instance, there is the class diagram available which shows the fundamental entities within the model. Furthermore, the composite structure shows the composition and structuring of these fundamental

entities including their interactions by use of communication channels. Furthermore, diagram types like the state machine and the activity diagram are available to visualize the dynamic aspect of a system definition in the model repository.

To summarize, UML diagrams only focus on specific viewpoints on the same system. In contrast, the XMI representation of the model represents a (possible) concrete syntax for the abstract model repository including diagram information. This representation is used to generate a valid SDL system description in SDL/PR, the textual representation of SDL (with the introduction of SDL-2000 now called *SDL/CIF level 0*).

When the UML CS profile is imported into a modeling tool its stereotypes are not required to extend each modeling element [WH06]. If a stereotype is defined as being *required*, the metaclass it extends cannot be instantiated without the stereotype extension. Using optional stereotypes allows combining several different views and modeling techniques with the profile itself. For example, use cases and deployments as well as different composite structure diagrams can be present in a UML CS model. However, the system definition always starts with a classifier that is extended by the stereotype *system*. This stereotyped classifier is the root of the system decomposition tree. Contained elements and agents also need the appropriate stereotype extension. Otherwise, they are not considered to be a part of the overall system specification.

6.1 Architecture

The architecture describes the static structure of a system. As an architectural description can involve a huge amount of entities, the UML CS profile offers several constructs to structure a description. All entities are subsumed into an *agent* concept. An agent is a general structural entity that offers the capability to decompose hierarchically into further sub-agents. In essence, it can be seen as a container for other agents. Therefore, it is allowed to place several agents within other agents. This cannot be done arbitrarily. There are several restrictions present having some semantic impacts (see Section 6.1.1.4). The available agent types in the UML CS profile are:

- *system*: top level, outermost construct, unique, non-empty
- *block* (optional): shall be contained in the system or in a block, non-empty
- *process*: shall be contained in a system, block or process

In addition, an agent may define operations:

- *operation* (optional): can be placed anywhere

Figure 40 shows the generalization hierarchy of the agent and operation concept. A description of a behavior can be done by a state machine or an activity. Both are specializations of the UML Behavior metaclass. An agent can specify a behavior by means of a *classifierBehavior* association. However, a block and a system must not have such a behavior specified. An operation is a BehavioralFeature which also specifies a behavior by means of its method. That is, the actual behavior of an operation is a method that is described by means of a Behavior metaclass specialization. As noted, a UML Behavior can be described by means of a state machine or activity. In the UML 2, a behavior can also be described by means of an Interaction. However, Interactions are not a supported behavior description in this profile.

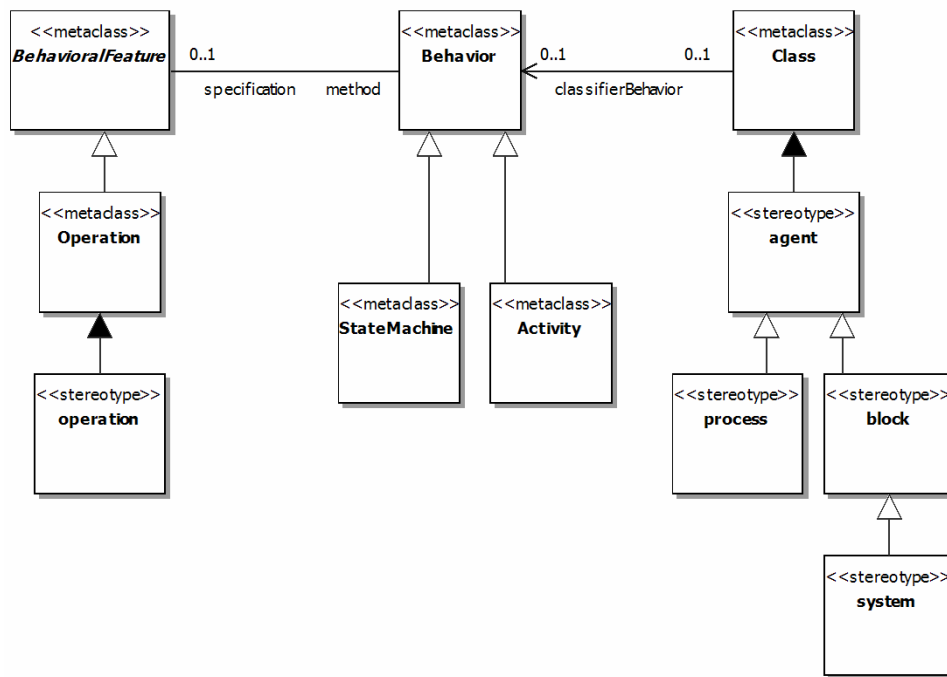


Figure 40: UML Metaclass Overview of the Agent Concept

6.1.1 Agents

Analogously to SDL-2000, the concept of an *agent* is introduced. An agent can be a system, or a block or a process. An agent may define variables, methods, state machines and further inner agents.

6.1.1.1 SYSTEM

The complete description is a description of a *system* agent which is a specialization of a block agent with some additional restrictions. A description shall contain exactly one system and the system shall contain at least another agent. A system specifies the boundaries of the known, specified domain. A system lies on topmost level of the structural hierarchy provided by this profile. The area outside of the described system is the *environment*. It is not specified what for a device is present beyond of a system and how its actual behavior is defined. However, the system description can communicate with the environment by means of signals. Thus, it is expected that the environment behaves in way that can be described by UML CS agents. This includes the ability to receive signals and to send signals including parameters with data types.

A system is a non-empty set of blocks or processes. These instances may communicate with each other. While not providing any details about the block's internal behavior, this high-level view of the system gives a first impression of the general architecture of the system that has been specified.

6.1.1.2 BLOCK

A block agent is a container for process or block agents. A process agent itself can only contain other process agents. A possible decomposition of a system into blocks and processes is shown in Figure 41. Only a process agent is able to execute a behavior directly and therefore may define a state machine and methods. In addition, the other agents can perform an observable, accumulated (or emergent, in UML terms) behavior. However, this behavior is caused by embedded state machines definitions by means of a process only.

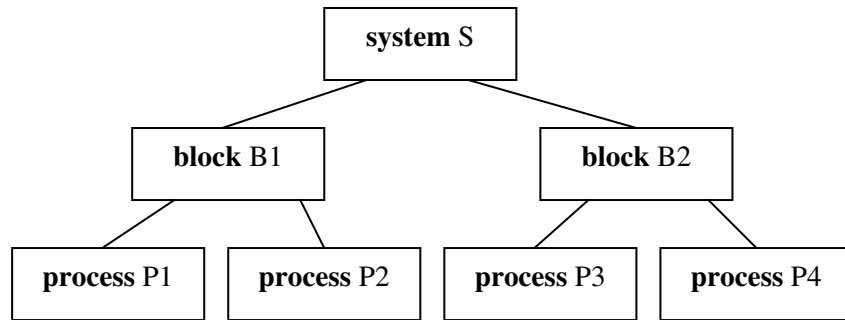


Figure 41: Hierarchical Decomposition of a System

A block is a structuring element that does not imply any physical correspondence with the target platform. Each block may contain a substructure of blocks to any depth or process sets. This decomposition of block in multiple sub-blocks can be used if it is beneficial for the internal structuring of this block. A block must not own variables although signals, operations and constant values are allowed.

6.1.1.3 PROCESS

A process agent has to define a state machine. Furthermore, several state machines can be contained in a state machine. However, there is only one state machine which is executed upon instantiation of the process. The other state machines can be invoked by the main executing state machine, e.g. on a transition or they can be invoked through the invocation of a method.

In fact, a process has one or more instances running in parallel and independently from the others. Variables declared in a first instance are also available in a second instance. There are two numbers available for specifying the total amount of instances running: The *lower* and *upper* multiplicity value. The lower value indicates the initial number of instances when the system is started; the upper value indicates the maximum number of instances that are allowed to run in parallel during execution of the system.

A Pid is a system unique identifier that unambiguously specifies an agent instance within a system. Each process instance contains four implicit local variables which are of type *Pid* (abbreviation for *process identification*):

- *self*: contains the Pid of the current agent instance.
- *sender*: contains the Pid of the agent instance which sent that last signal input
- *parent*: contains the Pid of the instance that dynamically created this current instance. It is a null value if this instance has not been created, but already exists at system start.
- *offspring*: contains the Pid of the last instance that has been created by this current instance.

These four variables are sufficient to identify each process instance. For instance, this is useful as it is possible to target a specific signal to a distinct Pid. By default, an agent has exactly one input queue where signals are placed in the order of reception. This behavior can be changed which will be explained in the next sections.

6.1.1.4 EXECUTION SEMANTICS

An agent container has a different execution semantics impact on its inner agents. The active elements (process agents using state machines) within a block agent execute *concurrently* and asynchronously. The active elements within a process are executed in an *interleaved* manner with atomicity at the transition level. In reality, also the inner agents of block agents are serialized if only one single host processor is available on the target platform.

6.1.1.5 SCOPE OF DECLARATIONS

A declaration is used to define data types, signal, variables and more. In UML CS, a declaration is visible to the current agent and to its inner agents. For example, a signal defined in a system is visible in the whole system with all its contained agents and entities. In contrast to signals and constant values, variables shall only be declared within processes. Declaring (global) variables in a system or in a block is not allowed.

6.1.1.6 LOCAL OPERATIONS

A method is the description of a behavior of an operation. In SDL, the corresponding construct is a procedure. Methods allow the parameterized re-use of certain code blocks. A method can be implemented by means of a UML Behavior. That is a state machine or an activity. The operation is only visible to the agent that defines it and to all sub-agents. The following Figure 42 shows an example of a local operation definition which can be identified by the visibility modifier *protected* of the *FindRoute* operation. This operation defines a formal parameter *pckDestination* of type *Pid*. Furthermore, parameters can also be concretized according to their direction (*in*, *inout* or *out*) or whether one of them represents the *return* value of a method.

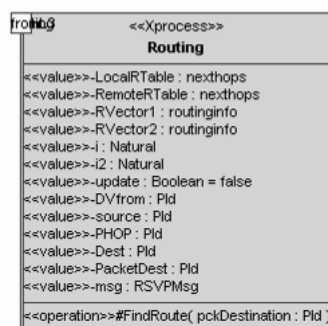


Figure 42: Local Operation Definition

6.1.1.7 REMOTE OPERATIONS

By default, an operation declared in an agent is only visible to this agent and to all its inner agents. Remote operations allow calling a method that accesses the context of a process instance from another process. This call must not necessarily be done within the same block. For example, this is a way to retrieve the value of a variable declared in another process, or to modify a value of a variable declared in another process.

An operation can be declared being a remote operation by assigning another visibility than the *protected* default. In the UML, there are several visibility modifiers available:

- *public*: the operation is visible to the whole system

- *package*: the operation is visible only to the package agents in which it is defined but not to agents which are not part of the package
- *protected*: the operation is visible only to the agent in which it is defined and to the inner agents. This is the default visibility of an operation.
- *private*: the operation is only visible to the agent in which it is defined. It is not visible to any other agent or inner agents.

6.1.1.8 EXTERNAL OPERATIONS

By default, operations of an agent have *protected* visibility. That is, the operation is only visible to the owning agent itself and its inner agents. However, operations can be accessed from external agents when the visibility modifier of the operation enables this. For this, the operation has to be the visibility kind *public* or *package*. Then an *operationCall* (see Section 7.4.6) can be executed which synchronously calls the operation and executes its behavior in the owning agent. The execution of the calling agent is resumed when the operation's behavior is finished. Operations having no appropriate visibility cannot be called.

6.1.1.9 STRUCTURING OF A SYSTEM

As already noted, a system can be decomposed into sub-blocks and processes. A block can contain another block or process. There may also be multiples of them present in a block. In Figure 43, the class diagram on the left and composite structure diagrams on the right show the internal structure of the class *RSVPEnv*. This class is extended by the stereotype *system* thus indicating that this is the top level of the system description tree. It is a feature of the composite structure that it gives an overview of the internal composition of an agent that is instantiated together with its contained agents. This implies when the system *RSVPEnv* is instantiated all containing agents are instantiated as well and the system is ready to be executed.

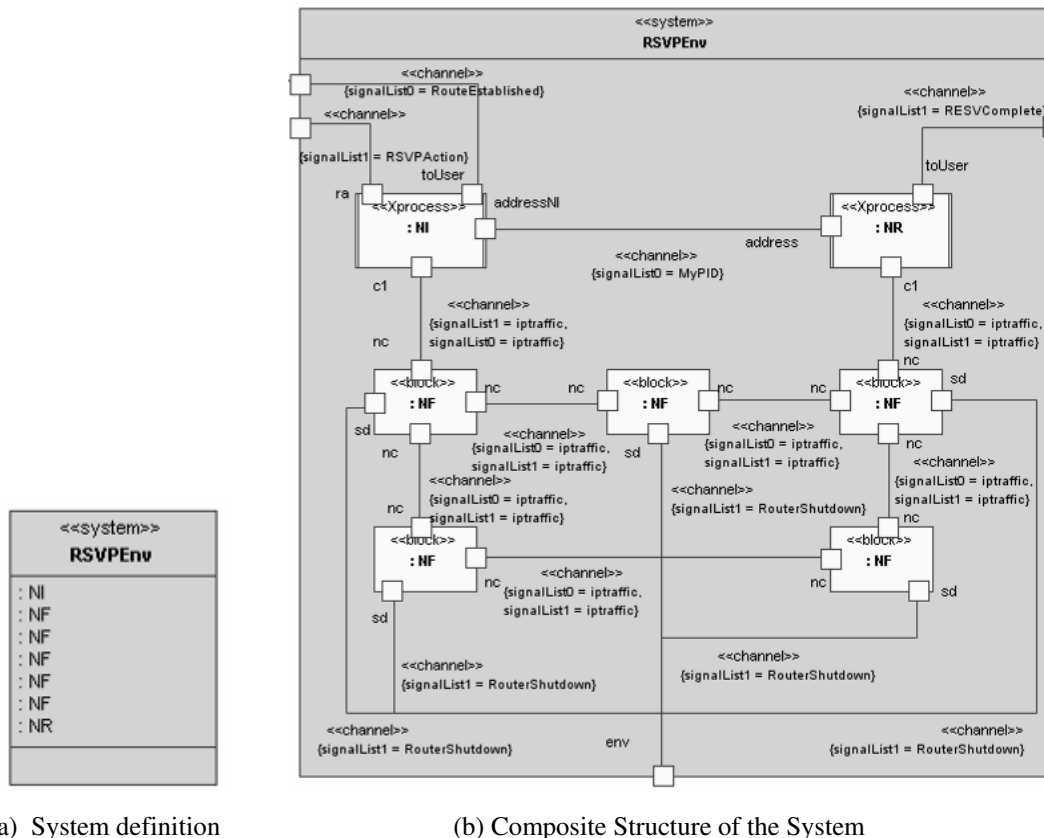


Figure 43: System with Blocks

This system shows several block instances labeled *NF*. The *NF* block type also contains several processes. However, these processes are encapsulated and thus hidden from the top-level system viewpoint. This allows a level of abstraction of the concrete implementation design.

6.1.2 Communication

The execution of UML CS models is driven by events. Events are a specification of a kind of an occurrence, for instance, reception of a signal or modification of a variable. In general, an event is the direct or indirect consequence of a behavioral action. An important criterion is that an event can be observed. There is no passage of time between the occurrence of an event and its observation.

A *signal* is a specification of an abstract information flow which can *trigger* some behavior. Two kinds of signals can be distinguished: *asynchronous signals* which represent asynchronous communication between two agents. For such an asynchronous signal, there can be three different associated types of events identified: the sending, reception and consumption of the signal. In contrast, a *synchronous signal* – which can be conceived as a remote procedure call (RPC) – models a synchronous communication between two agents. For this kind of communication, there can be further associated events identified such as invoking a remote reaction or returning from the reaction to the signal.

Events are substantially different from messages and actions. The execution of an action may involve several events to be observed. As noted, the action of sending a signal leads to the events of the sending and reception of this specific signal. This signal will be represented by a message. It is the concrete instantiation of a particular signal or remote call being conveyed from one agent to another. The same message can be involved in multiple events both times when it is sent or received.

The UML distinguishes between active and passive classes. Active classes have their own thread of control and declare the existence of a state machine which specifies the internal behavior of the class when instantiated. Passive classes define objects which do not have an own thread of control. Invocation of owned operations is carried out immediately. This can possibly raise race conditions. Passive classes are used for communication between active classes. Passive classes define signals or data types for instance. A signal may contain additional data variables. However, both types of classes support object-oriented features like inheritance and encapsulation.

6.1.2.1 SIGNAL

A signal is an abstract specification of send requests communicated between agents. A signal can also define attributes that are part of the transmission object. So, the data carried by a send-request are represented as attributes of the signal. This data was passed to the signal by the send invocation event that caused that request. A signal event can invoke a trigger which results in a reaction of the receiver in an asynchronous way and without a reply. The sending agent of a signal will not block waiting for a response, but continues execution immediately.

A signal is an attribute of an agent. An agent has to declare its ability to receive specific signals. Without such a specification the model is ill-formed. This declaration is an interface on an agent's port. The union of all provided interfaces of the agent's ports defines a set of all signals the agent is able to receive. Nevertheless, an agent itself is also able to restrict the reception of a signal to a specific port.

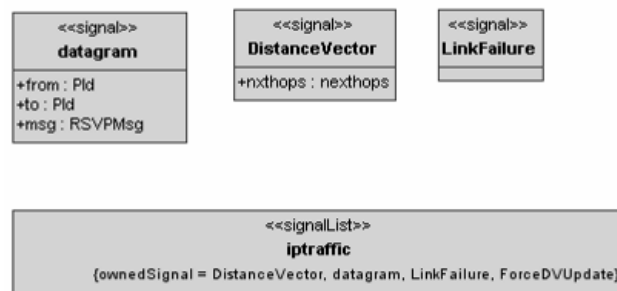


Figure 44: Signal and Signallist Definitions

Figure 44 shows some possible definitions of a signal. The signal *datagram* on the left has three attributes: The *from* and *to* attribute are of type *Pid* (process identification type) and the *msg* attribute is of type *RSVPMsg* which is a composed data type. The definition of *RSVPMsg* is not shown in this excerpt. The signal *DistanceVector* has only one attribute *nxt hops* that is of type *next hops*. This data type is also not shown here. The *LinkFailure* signal has no attributes defined. Note that the attribute names are only depicted for informational purposes and impose nothing on the system. Only the type of the attribute is important for the definition of signals.

Also shown is a *signalList*, a specialized composite type of a signal. A *signalList* is a set of signals gathered under a unique identifier. This identifier can be used as shorthand if several signals are often used together, e.g. on a channel.

6.1.2.2 CHANNELS

The communication between agents is accomplished by the *channel* concept. Channels activate a communication path between both connected entities. A channel cannot directly be connected to an agent, because agents define interaction points where communication items have to be conveyed. The

type definition of an interaction point is done by a *port*. A channel can be specified as being a delaying channel. For this, the channels tagged value *delay* has to be set to true. A delaying channel contains a FIFO (First-In First-out) queue used to delay the signals. Delaying a signal may especially be useful in protocol validation by simulation, for example. A channel without a delay does not have a FIFO queue for signals.

A channel can convey signals either in one direction or in both directions. This depends on the *signalList0* and *signalList1* tagged values which are of type signal with multiplicity 0..*. For the corresponding *connectorEnd* attribute such a *signalList* defines the signals that can be conveyed to these specific endpoints. It is an ordered set of the channel's endpoints. As a channel cannot connect more than two endpoints there are only two *signalList* tagged values available.

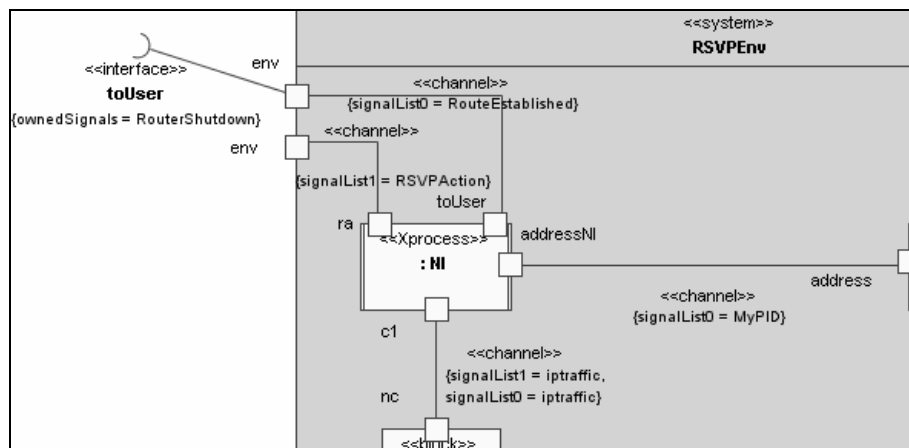


Figure 45: Definition of Channels

In Figure 45, the excerpt of a composite structure diagram of a system gives an example of a possible channel definition. For this, the tagged values are also shown in the diagram. For example, the channel connected to the ports *addressNI* and *address* has a *signalList0* with the signal *MyPID* assigned. This indicates that the port, which is connected as the first *connectorEnd* attribute of the channel, can receive the signal *MyPID*. The port on the other *connectorEnd* does not receive any signal via this channel. Unfortunately, stating which end of the channel is the first one and which one is the second is only declared in the model repository and not depicted in the diagram.

6.1.2.3 PORTS AND INTERFACES

A port is the type definition for an interaction point for the communication between an agent and the attached channel. All communication is performed through ports and their associated channels. When a signal is directed through a port, the signal always traverses the following sequence of entities of a system: agent, port, channel and target port and target agent. A port defines the signals that can be received from other agents and those that can be received by other agents. Furthermore, a port defines the operations that are implemented in its owning agent and the operations that are required to implement in other agents. This enables Remote Procedure Calls (RPCs) which have been discussed in the previous sections 6.1.1.7 and 6.1.1.8.

A port refers to two interfaces: The *provided interface* and the *required interface*. The provided interface publishes the internal reception availability, a signal and the implementation, an operation, of the owning agent. For example, a provided interface, which owns the signal *sig1*, declares that its owning agent is able to receive the signal *sig1*. The required interface announces that this agent requires other agents able to receive a signal or operation call which may be sent by this agent through this specific port.

By default, if a port is defined on its owning agent type, this port has also to be instantiated on all instances of the agent. For example, if an agent *ag* has the ports *portA* and *portB* defined, both instances *agent1:ag* and *agent2:ag* have to have two instances of *portA* and *portB* attached. All these ports have to be connected by means of channels.

DYNAMIC PORTS

An exception is the *dynamic port*. Dynamic ports can be left unconnected and they are considered non-existent in this case. A port being a dynamic port is indicated by its tag definition *isDynamic*. Dynamic ports define a *dynamic port set* which are identified by the identical name and by the *isDynamic* tag definition of all ports set to true. Each element of this set is addressed by a trailing index. The total amount of available dynamic ports is determined by the tag definition *instances*. Following Figure 46 is an excerpt of a composite structure diagram and illustrates the handling of dynamic ports:

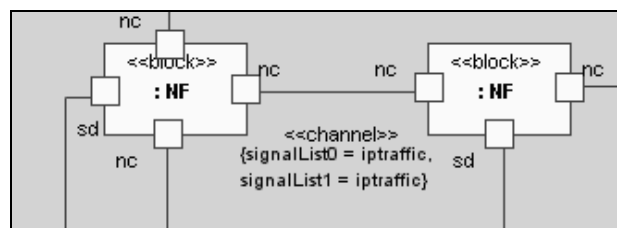


Figure 46: Dynamic Ports attached to Block

As shown in this figure, there are two (anonymous) instances of the *NF* block. The *NF* block owns two different ports – the port named *sd* and the dynamic port set *nc*. The latter one can be identified by the duplicate names which is not allowed for normal ports. Additionally, the tag definition *isDynamic* of all the *nc* ports is set to a true value. On the block instance on the left, there are three instances of the *nc* port. The block instance on the right has only two of the port instances attached. The third one is not connected so it is removed internally before execution of the system. As noted earlier, the addressing of a dynamic port is done by a trailing index. This means, the block on the left addresses the elements of the dynamic port set by *nc(0)*, *nc(1)* and *nc(2)*. The *instances* attribute of the *nc* port result in the natural value of three as there are three dynamic port instances connected. This implies that the index of the port must not be equal or greater than the reported amount of port instances (as counting begins with a zero index). The block agent on the right can only address the ports *nc(0)* and *nc(1)* while the *instances* attributes reports two instances. However, it is not possible to assert the index value to a specific port. In fact, the index value of a port might vary from each system start. It is up to the corresponding agent to establish neighboring process identification and the port index association.

6.1.3 Generalization

UML CS supports generalization of agents. That is, a specialized agent can inherit all features and properties of its general agent (parent). Furthermore, the specialized agent is allowed to add new properties, states and behavior. Note that it is currently not possible to re-define certain properties of an agent (see Section 7.2.8 for rationale). Therefore, it is only allowed to add new states or transitions.

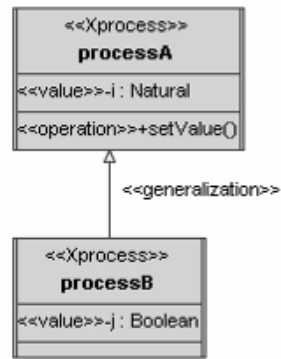


Figure 47: Generalization of Agents

Figure 47 shows an example of a possible generalization between agents. In this particular example, the agents are processes. *ProcessB* inherits all properties from its general process *processA*. This includes variable *i* and the operation *setValue()*. In addition, the *processB* declares another variable called *j*. This variable *j* only is visible to the *processB* and not to the *processA*. The contained state machines are also inherited to *processB*. The re-definition of the state machine is not shown.

6.2 Behavior

6.2.1 Behavioral Semantics of UML 2

A *process* is the only instance in a system that directly executes a behavior. Observable behavior of the other agent types is the result of the aggregated behavior of processes. Any behavior is the direct consequence of at least one agent performing some action. A behavior describes how the internal configuration of these objects changes over time. There are two kinds of behaviors defined in the UML 2 [Sel04]: *emergent behavior* and *executing behavior*. An *executing behavior* is performed by a single agent and is the description of the behavior of this agent. An executing behavior is directly caused by the invocation of a behavioral feature of that agent or by its instantiation and is running within the scope of the agent gaining access to structural features. However, it is a consequence of the execution of an action done by some related agent. *Emergent behavior* results from the interaction of one or more participant agents. If the participating agents are parts of a composed agent, an emerging behavior can also be seen as the aggregated behavior, thus describing the behavior of the container agent. An emergent behavior can result from the executing behaviors of the participant agents.

A process agent can communicate with other processes or with the environment. A process is a structural representation of the code that will be executed. In UML CS, a process' behavior is specified by a communicating enhanced finite state machine. A behavior is user-defined while the underlying activities are defined by the UML. A behavior can be specified by means of activities. A finite state machine has a finite set of states and one special start state called *initial state*. This is where the execution begins. The finite state machine waits in a state – the *active state* – until an event occurs. This event can trigger a *transition* to be executed. This, in turn, results in the current state being left and a subsequent state being set as the new active state. Which state is left by the occurrence of an event and which state is entered as the new active state is defined by a set of *transitions*. A *transition* defines for a state which state is the subsequent one if a certain event occurs. Additionally, the transition may specify expressions, called *guards*, which have to be satisfied and evaluate to a true value to enable the traversal of this transition. Therefore, the subsequent state of the current state defined by a transition will only be entered if the specified event has occurred and the guard

expression has evaluated to true. If any of these two conditions is not met, the transition is not taken. A possible event is the reception and consumption of a signal which has been received from another process. Besides, a transition may also specify a behavior that has to be executed if the transition is taken. The possible behavior is discussed in a later section. Nevertheless, it is important that this behavior might include a sending of a signal to other processes. This feature extends the finite state machine to a communicating finite state machine.

Some algorithms can only be calculated by a state machine if a huge amount of states is being used. This is called *state explosion*. To avoid this complexity, *variables* can be added to the state machine that stores values for calculation in these variables. This may help reducing the amount of states necessary within state machines. This renders the communicating finite state machine to a communicating enhanced state machine. Note that each state machine (process) in UML CS has an implicit message queue to receive messages. Therefore, it is possible to have several process instances of the same process definition running concurrently without any interference on signal reception.

A process is a specialized class of an active class. When a process is modeled, it is instantiated from this stereotyped class definition. Thus, all processes have to be instantiated from a process class definition. Contrary to SDL, a direct declaration and implicit instantiation of a process is not possible. The corresponding definition type of a UML CS process in SDL is a *process type*.

6.2.2 State Machines

A state machine is one way to define the behavior of an active entity. Currently, most tools only support the state-centric state machine view in state machine diagrams. In this type of diagrams, the state machine only consists of states and Pseudostates (i.e. decisions, terminate, initial) and transitions. The actions are included by means of activities. However, as processes execute a behavior on instantiation, a state machine is used for description.

As state machine-based concurrency is not supported, a state machine must contain only one region and one start state. As shown in the example in Figure 48, there is one start state. This start state is connected by a transition to the state labeled *idle*. During the transition, there is an activity named *init* defined. This activity is not shown in the state machine diagram. It is only shown in a separate activity diagram. The behavior on a transition is implemented by means of an activity.

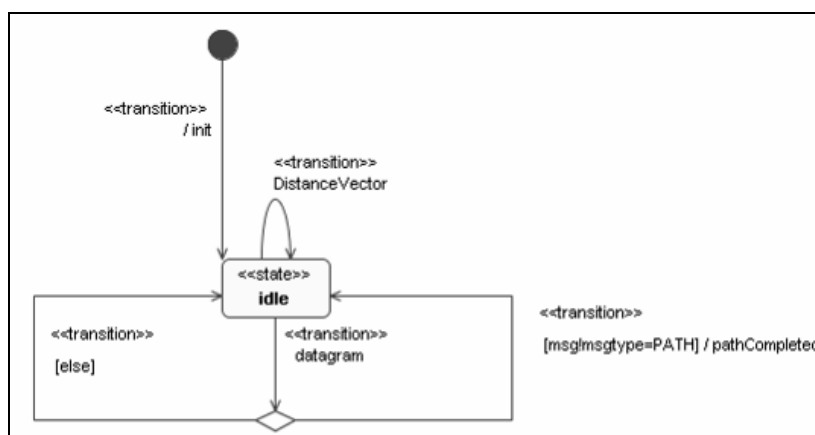


Figure 48: State Machine with one State

6.2.2.1 STATE

In UML 2 state machine diagrams, a behavior can be specified within a state or transitions. In a state, behavior can be specified as *enter*, *exit* and *doActivity*. When a state is entered, the behavior specified as *enter* is executed. While the state machine remains in this state, the *doActivity* is executed. When an outgoing transition from this state is being triggered and the constraints are fulfilled the *exit* behavior is executed.

Nevertheless, in this profile actions within a state are not allowed. A behavior can only be specified in a transition as an *effect activity*. Following Figure 49 shows an excerpt of the relationships of the *Transition* metaclass in the package *BehaviourStateMachines*.

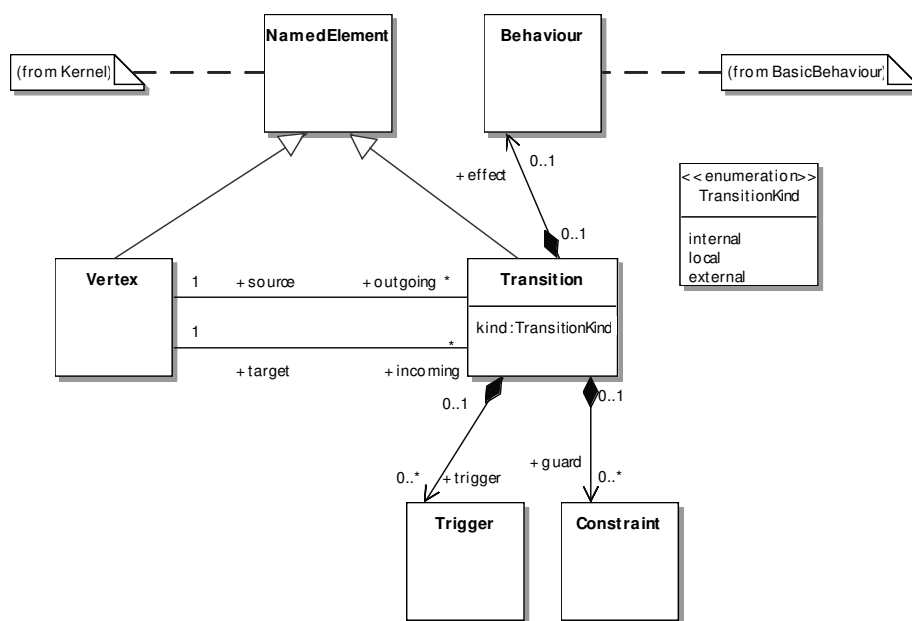


Figure 49: Excerpt of Transition Metaclass in the UML metamodel

The default notation for a transition is defined by the following BNF expression as defined in [OMG06]:

```

<transition> ::= <trigger> [',' <trigger>]*
                ['[' <guard expression> ']'] [ '/' <behavior expression> ]
    
```

A state can declare triggers to be deferred to the subsequent state if there is no transition available to be triggered. To define a list of deferrable triggers, the state's *deferrableTriggers* attribute is available.

In addition, a list of states can be specified. This is shorthand to associate the same transitions with several different states. A list of states has to be indicated by means of the *isStateList* attribute. The state names must be separated by commas. The state list can be notated inverted by means of an asterisk with the state list bracketed.

6.2.2.2 TRANSITIONS/CONTROL FLOWS

A transition connects a (pseudo-)state with another (pseudo-)state. A transition is selected if its source state is active, the optional guard being a Boolean expression is satisfied and the transition is triggered by an event. Two other factors influence the transition selection. At first, the prioritized transitions are

evaluated. If none of the prioritized transitions can be selected, the non-prioritized transitions are evaluated. As a signal can have a priority, the transition with the triggering signal of highest priority is selected. A trigger must not be specified if the source state is a pseudostate.

A control flow in an activity is only used to specify the possible control flow within an activity. A control flow is not allowed to specify triggers. Guard expressions are only allowed if the source node is a decisionNode.

6.2.3 Signals

Objects communicate because of state machine instances sending signals. State machines can send signals to establish the communication and to synchronize the behavior of objects. A signal is an abstract message that can carry data used by the actions of the recipient's state machine. The signaling is asynchronous. Once the signal is sent, the sender instantly continues with its execution. Separately, once the event is detected by the receiver a transition is made and the receiver may execute a behavior. The receiver does not reply to the sender though it may choose to send another signal.

The UML 2 allows an alternative notation for signal reception, signal sending and other actions. This notation is depicted in the following subsection, and this notation is the preferred notation in this profile. However, this notation is rarely supported in current UML modeling tools.

6.2.3.1 SIGNAL RECEPTION

For a signal reception, the trigger symbol is shown as a rectangle with a triangular notch in one of its sides. The signal reception symbol – an *input* – represents the trigger of the transition. The textual trigger specification is denoted within the symbol. The transition may also define a guard and is given as a Boolean expression within the input icon as shown in following Figure 50.

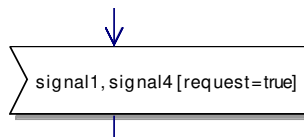


Figure 50: Alternative Notation for Transition Trigger and Guard

A compound transition is an ordered, complete series of forward directed transitions between two states connected by these transitions and potentially interrupted by pseudo states. The input symbol is always the first symbol in a compound transition between two states. There can be only one of such symbol within a compound transition. Signals can also have an assignment specification that assigns the data values – which have been conveyed by this signal – to local variables. Signals may have a priority. That signal having the highest priority is consumed first. If there are multiple signals in the input queue having identical priorities, this signal is selected that has arrived first. A transition is only selected if its guard is satisfied (or enabled). Afterwards the trigger is evaluated, thus not removed from the trigger input pool. That implies that the transition may be taken although the event has occurred before the change event of the guard expression (note that this behavior is assumed to comply with the UML semantics – however, other opinions on this specific behavior are known).

INPUT VIA/FROM

A triggering event can be constrained depending on the port which received the signal or from which process identification – which addresses the sending process – the signal has been sent. For both constraints, the attribute *port* (from the SignalEvent which has invoked the trigger) or *sender* (from the received Signal) are available.

6.2.3.2 SIGNAL SENDING

Signal sending is an action that has a special notation, named *output*. When an output is executed, the given signals are created and their instances are sent to the target agent passed. The target can be addressed implicitly or explicitly, by either a specific port or process identification (Pid). If there are multiple potential targets available, an arbitrary one is selected. If actual parameters of the signal are defined, the parameters are shown within the symbol. Within a given compound transition, the signal-sending symbol must succeed the trigger symbol if the latter exists. The signal sending symbol is mapped to a `SendSignalAction` extended with the stereotype *output*. In the case that the modeling tool does not support the actions package, the details of the output action can alternatively be defined within the body of the behavior instead of using the output symbol. It is possible to have multiple output nodes on a compound transition path.

OUTPUT TO/VIA

Output has an attribute *target* that specifies the InputPin of the receiver object Pid to which the signal is to be sent. In addition to the graphical icons, the textual notation for output is shown in the next Figure 51.

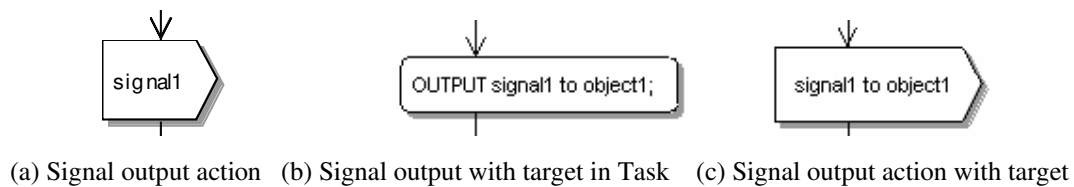


Figure 51: Different Notations for Signal Output

Besides of the direct addressing via a Pid, the target can also be chosen by sending the signal via a specific outbound port using the *via* attribute. While a target addressing of the output cannot be ambiguous this can apply to *via*. In this case an arbitrary target is selected.

6.2.4 Other actions

An action symbol is drawn as a rectangle and contains a textual representation of the complete action represented a transition (see Figure 52). This symbol has to immediately succeed the signal receipt symbol and connect to the subsequent state. It can be defined at most once within a compound transition. The action sequence symbol is mapped to an opaque action with its *language* attribute set to "SDL". It is also possible to map the action symbol to an activity with a sequence node containing instances of actions, depending on whether it represents one or multiple actions, respectively.

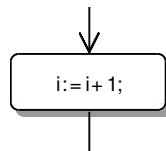


Figure 52: Alternative Behavior Description in Action

6.2.4.1 RANDOM

The UML CS profile expects that a suitable random library is available which allows implementing the random functions. The list of available random function is described in Section 7.5.

6.2.4.2 TASK

The action language for UML CS is SDL in its textual representation (CIF level 0, formerly SDL/PR). For this, an *OpaqueAction* is available extended with the stereotype *task*. This *task* box allows the specification of textual language statements.

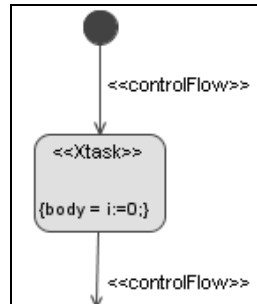


Figure 53: Task Box

A *task* box is shown in Figure 53. The body attribute is a set of strings that allow specifying SDL language statements. When SDL is the target mapping language, these statements are directly mapped. In this case, the value zero is assigned to variable *i*. Note that the visible stereotype *Xtask* is a special adapted stereotype for the modeling tool. It is used because this tool does not offer a graphical representation for an *OpaqueAction* – a UML metaclass that directly mapped to the target language.

6.2.5 Control Flow Statements

6.2.5.1 CHOICE/DECISIONNODE

A *choice* (for state machines) or a *decisionNode* (for activities) is a dynamic conditional branch used to define a possible control flow of the execution. A choice consists of a common *question* and several *answers* on the outgoing transition. This condition is depicted as guards on the outgoing transitions of the choice node. A pre-defined *else* guard is available which yields a true value if and only if all other guards are false. There can only be one single *else* guard being connected to a choice node. Figure 54 depicts a *decisionNode*.



Figure 54: decisionNode with Conditions

6.2.5.2 MERGE/MERGENODE

A *merge* (for state machines) or *mergeNode* (for activities) is used to link several transitions or control flows together. If the subsequent transition does not define any activity, it is only used for graphical intelligibility and is optional. These nodes can also be omitted and the control flow can be connected to the subsequent node directly. Figure 55 shows an example of a merge node in a state machine. The notation within an activity is likewise with the exception that an activity is not allowed to define any further activity on a control flow.

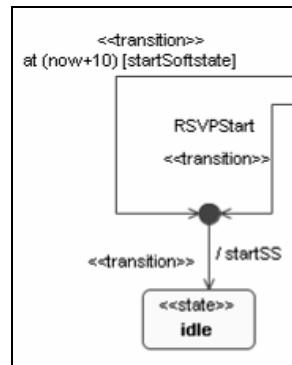
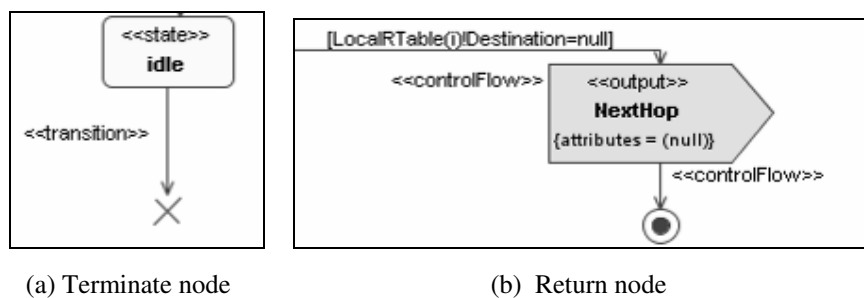


Figure 55: Merge Node

6.2.5.3 TERMINATE AND RETURN

A process can be terminated by means of the stop node. The process executes no further activity after this point. A state machine can also describe the behavior of an operation; this is the method. A method can only return the control flow to its caller and cannot terminate a process. The same applies to the description of a behavior by means of an activity. In addition, an operation of a data type can only be described by an activity (and not by a state machine). For returning the control flow to the caller and resuming the execution, the return node is available. Both node types are shown in Figure 56.



(a) Terminate node

(b) Return node

Figure 56: Terminate and Return Node

6.2.6 Timer

A timer is a specialized form of a signal. A timer can be scheduled to send a signal to its owning process after a specific amount of system time has passed. The timer can also be re-started or disabled.

6.2.6.1 TIMER DECLARATION AND STATEMENTS

A timer is declared in the same way as a signal. There are several time control statements available. Besides of the proposed graphical notation – which is a hourglass symbol – a textual notation is supported which is equivalent to the SDL syntax. The supported notations for starting a timer is

```
'set(' <timer identifier> ', ' <time expression> ')' <semicolon>
```

To disable a timer the following statement is available

```
'reset(' <timer identifier> `)' <semicolon>
```

Finally, a timer can be checked whether it is still active and scheduled for a timeout. For this, the following statement is available:

```
'active(' <timer identifier> `)' <semicolon>
```

6.2.6.2 SOFT STATES

A soft state is a state that has an implicit timer running. This timer is capable to trigger a transition when a certain time duration has passed. The notation for such a time trigger is *at(<time expression>)* or *after(<duration expression>)*, see Figure 57 for an example. Note that there may be more than one time trigger deriving from the same state. The soft state timers are implicitly re-started when the state is re-entered. If another transition is triggered, the soft state timer is implicitly deactivated.

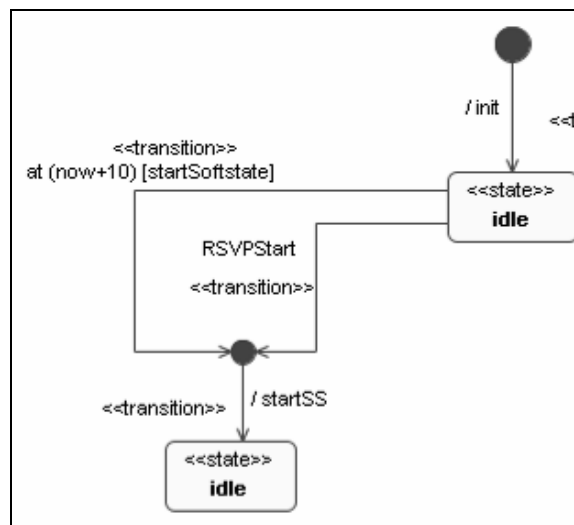


Figure 57: Notation for Soft State

6.3 Data

The UML CS profile supports almost the same data concept as it is implemented in SDL-2000. There are only minor deviations. All pre-defined data types that are available in SDL-2000 are also available in this profile.

6.3.1.1 COMPOSITE AND PRIMITIVE DATA TYPES

In addition to the primitive types, also composite types like structures (*struct*) and unions are available. Primitive data types are defined by means of literals and operations which define a specific behavior on data types based in these literals. Structures compose several data types into one cohesive package. Each of the elements of the structure (*field*) is directly accessible and can be modified without interfering with the other. Unions are similar to structures with the difference that only one single element (*variant*) can be accessed at the time. If a value is assigned to a union variant, all other variants are overwritten.

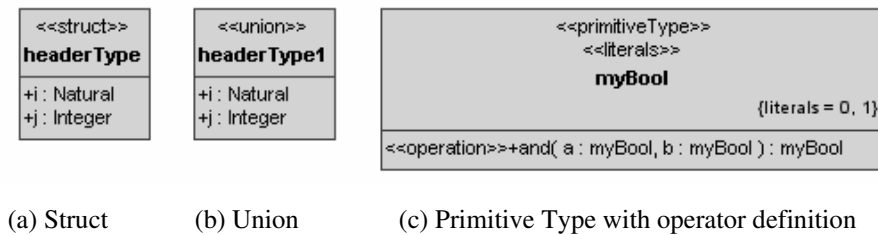


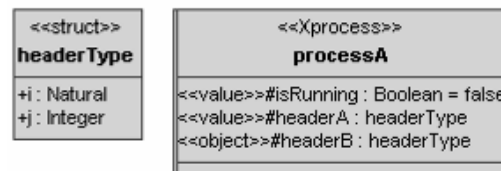
Figure 58: Definition of Primitive and Composite Types

Figure 58 lists three types of data type definition. The data type on the left is a definition of data types being extended with the stereotype *struct*. This indicates that the owned attributes – named *i* and *j* – are field entries of the composite type. The data type in the middle is a *union* composite type defining the same data types. This is indicated by the applied stereotype *union*. The data type on the right is a primitive data type that defines the value domain, the literals 0 and 1 and specifies an operation *and* which operates on these values. This operation takes two values of type *myBool* and returns a result of type *myBool*. The implementation of the operation is not shown here. However, the operation's method shall only be described by means of an activity.

6.3.1.2 VALUE AND OBJECT TYPES

Furthermore, there are two different types of data type semantics available: *value types* and *object types*. The difference lies in the handling of assignments. An assignment of a value-type implies that the value of the data type is copied to its target. In contrast, object-type assignments copy the reference value of the data type to the target data type.

Following Figure 59 shows an example of a structured data type including its declaration as a value and object type.



(a) Struct definition (b) Process defining a value-type and object-type

Figure 59: Declaration of Composite Data Type and Type Semantics

6.4 Summary

In this chapter, an overview of the UML CS profile has been given. This covered the architectural and behavioral concepts as well as timer mechanism and data types. This overview has provided a user's view of the profile which is not implicitly clear and visible from the following profile definition in the following chapter itself.

In the first section, the architectural concept has been explained. This is the agent concept that allows a hierarchical decomposition of a structural specification into smaller parts. The communication base and the available construct have been depicted and an informal description has been given. The second section has described the behavioral concept of the profile. The main theoretical formalism used for this is the communicating extended finite state machine (CEFSM). The graphical elements to define a finite state machine have been discussed and the communication and activity concepts were listed.

Besides of the available concepts in UML, the focus of the overview lied on the new language features described in Chapter 5. In the third section, the data type concepts and its capability to define new operators and composite types have been presented.

7 Profile Definition

The profile described in this thesis is driven by the formality, experience and intelligibility of SDL. However, this profile is not limited to SDL and it does not constitute a simple one-to-one mapping to SDL. In particular, there are several additional high-level modeling constructs available which cannot be specified in SDL such as an input via a specific port, from a specific agent or ports with an individual input queue. In addition, there are some language features available in SDL that cannot be specified with UML CS currently.

In this chapter, the stereotypes of the profile are semi-formally defined. The first section provides an introduction to the overall profile's design. The subsequent sections describe the conceptual parts of the overall profile: The second section describes the stereotypes that extend several UML metaclasses for the structural specification and description of a communicating system. The third section describes the stereotypes for the UML Behavior by state machines and activities. The fourth section describes the stereotypes for UML Actions. The fifth section defines the stereotype for using random operations; the sixth section provides the definitions of the stereotype extensions for timer definition and manipulation. The seventh section covers the stereotypes for data type definition. The stereotype definition consists of a syntax and informal semantics and additional information. The constraints of the stereotypes are defined by means of a formal specification language – the Object Constraint Language (OCL). This enables UML modeling tools to check automatically if these constraints are satisfied during a communication protocol's specification and description process.

7.1 Introduction

With the introduction of SDL-2000, SDL has reached a high level of object-oriented formal description power and expressiveness. Unfortunately, this was accompanied by high level of complexity as noted in [KLP+04, She05]. It is argued that several features available in SDL are only used by a minority of users. This becomes underlined as the tool support of SDL-2000 is still poor. Most tools are even not fully SDL-96 compliant. In contrast, several concepts for the modeling of next-generation telecommunication protocols are cumbersome to be described by SDL as already noted in the previous chapters.

Hence, instead of defining a new language concept from scratch for a UML profile, a mapping from the UML to SDL will be defined. The semantics of SDL have been formally described by means of the Abstract State Machines formalism. It has undergone several revisions and improvements since its initial release [Pri02]. The SDL semantics will be re-used for this profile. In most cases, this allows a mapping from a UML system to an SDL design specification. Based on the translated specification, the static and dynamic semantics of the UML model can be simulated and validated. In addition, the syntax has to be verified by the SDL tool, because textual SDL language expressions can be defined in several UML CS model elements. These expressions are (currently) not checked in a UML modeling tool. Hence, this allows using any SDL statement within a system description even if it is currently not supported by this profile. An overview of the specification and description process for communicating systems is given in Figure 60. Providing a mapping to SDL shows that a formal description of telecommunication systems is feasible by using the UML CS profile.

However, this approach currently constitutes the downside that in case of errors a direct correspondence between the erroneous language element in the UML and the detected error in SDL cannot be made. The reason is that some UML CS elements are grouped together in a single SDL

statement while others are decomposed into several SDL statements. This circumstance sometimes may only allow to give a rough estimate of the location of the error. In the future, however, modeling tools might be able to support UML CS directly without the need to map the UML diagrams to SDL first. This would allow a much tighter integration of the software engineering process covering the requirements, analysis and design and further to the implementation.

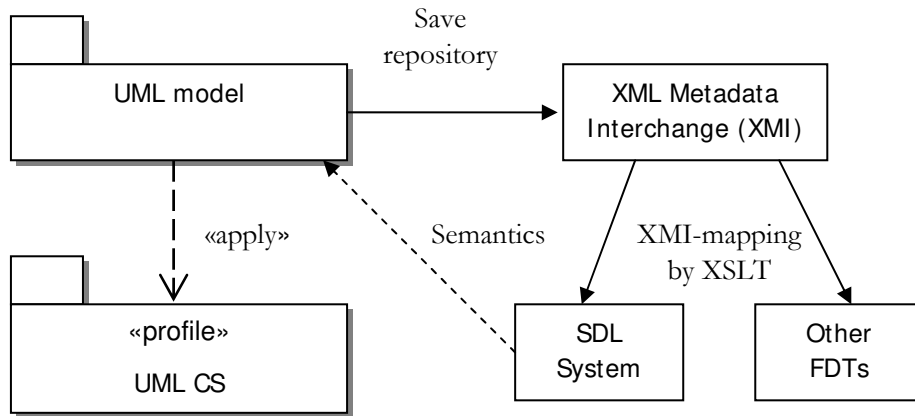


Figure 60: Specification and Description Process using UML CS Profile

As in SDL, UML CS introduces concepts (especially for larger systems) for structuring, behavior and data description. The basis of structuring is the hierarchical decomposition of system views and hierarchies of types. The basis for behavior description is a communicating extended finite state machine. Data description is based on types of objects and values. These are the three constituent parts of this profile and defined in the following sections in detail.

7.1.1 Reading the Profile Definition

The profile comprises of stereotypes that can define tags, tagged values and constraints. For clarity, each stereotype is defined by means of a table.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Port		9.3.11 Port
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	<i>TAGGED VALUES</i>
«port»	Port (from Ports)	inputQueue: Boolean = false
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context port inv: self.isBehavior=false inv: self.isService=true		The optional <u>inputQueue</u> specifies if inbound signals are queued by this port until the are consumed by a process.

Figure 61: Example of Stereotype Definition Table

An arbitrary example of such a table is shown in Figure 61. This table provides the following information: First, it gives a name to the stereotype that is shown in the cell labeled *UML CS Stereotype*. This unique name identifies this stereotype. Either a stereotype can extend a UML metaclass or it can specialize another stereotype. The base metaclass to which this stereotype refers to is shown in the cell *UML Node Type*. This identifies the modeling element (UML metaclass) that is extended. The metaclass or stereotype, which is extended by this stereotype, can be found in the cell *UML MetaClass*. In this example, the stereotype «port» is extended from the UML metaclass *Port (from Ports)*. The expression *from* shown in brackets denotes the package in the UML Superstructure where this metaclass is defined. The package qualifier is needed as some metaclass are incrementally defined in multiple packages by the UML package merge technique. Furthermore, the relevant UML Superstructure document's section heading of the extended metaclass is shown in the cell *Reference*.

Additionally, the profile proposes a graphical notation. This symbol is shown in the cell *UML Notation*. In most cases, further alternative notations are given that should be preferred depending on the UML tool's capabilities. If they are provided, they are not added in this table, but given in the description. Note that textual notations are specified in the Extended Backus-Naur Form (EBNF), described in Section 3.3.2. Most of the expressions, references and rule names refer to the EBNF defined for the concrete grammar in the SDL-2000 document [ITU02a].

For the definition of a stereotype, constraints and tagged values can be defined. The constraints are specified in OCL. The constraints, which apply to this stereotype, are shown in the cell *OCL Constraints*. In most cases, these constraints are underlined and explained by an informal description which can be found under *Informal Constraints*. Note that both types of constraints must not contradict. If such an event occurs, none of them has precedence over the other. This error must be corrected. Finally, the additional attributes defined by this stereotype are given. An attribute definition of a stereotype is named a *tag definition*, and their concrete assignment is a *tagged value*. Therefore, this can be found in the cell *Tagged Values*. In this exemplary case, the port stereotype defines an additional Boolean attribute *inputQueue* which is initially set to a false value. Each stereotype also has an informal semantics description. If the informal semantics description refers to a metaclass attribute or stereotype tag definition (attribute of a stereotype), this specific attribute name is underlined.

For clarity, there are auxiliary OCL operations defined which are re-used in the constraints of the stereotypes. The following OCL operation *isStereotypedBy* results in a true value if the given element is stereotyped by the specified stereotype classifier. Otherwise, it results in a false value.

```
UMLCS::isStereotypedBy(e: Element, s: Stereotype) : Boolean;
post: result=e.extension->exists(e | e.type=s)
```

The OCL operation *isRemote* checks whether two *NamedElements* are defined within the same namespace. This is used to check if an *operationCall* action calls an operation within the same agent or not, rendering it a Remote Procedure Call (RPC). This is also required to recognize access to local or remote variables. The operation *isRemote* returns a true value if both given elements acting as parameters are belonging to different namespaces.

```
UMLCS::isRemote(n1: NamedElement, n2: NamedElement) : Boolean;
post: result=n2.allNamespaces()->reject(n1.allNamespaces()->notEmpty())
```

The result of the OCL operation *random* is a unique, randomized String value. This feature cannot be expressed in OCL. However, the returned String value must never be identical to a previous value for the same system. This operation is required to assign an identifier to implicitly defined elements within a model.

```
UMLCS::random() : String;
```

Notice, all stereotypes use a multiplicity of 0..1 for extension. This implies that there are UML modeling elements allowed without having the appropriate stereotype extension. This is intended to allow a combined notation between UML CS models as well as other profile elements or non-extended UML model elements within the same model repository. However, if model elements are used within a UML CS model without having the appropriate stereotype extension, the model semantics is undefined.

In this chapter, there is a specific notation used for UML metaclass elements: References to the UML metaclasses are capitalized, e.g. Event, Transition or Trigger. Attributes of the metaclasses are written in lowercase letters in the text and are underlined.

7.1.2 Queue Disciplines

The invocation of a behavior by means of signals is a sequence of occurrences according to the UML common behavior: a signal may have an associated Event, in particular a SignalEvent that is a specialized form of an Event. If the SignalEvent occurs, it may invoke a Trigger. This Trigger can invoke a behavior, e.g. on a state machine's transition. According to the UML Superstructure document, events are placed into an input pool. The event is dispatched when it is taken from this input pool and causes a behavior, either directly or indirectly. It is removed from the input pool thereafter. It is a UML semantic variation whether an event is discarded if there is no appropriate trigger defined for them. In UML CS, this is specified as described in the state semantics in Section 7.3.3.

The UML input pool semantics for pending events is constrained in the UML CS profile. Currently, only SDL communication semantics is supported for signal events which is a single input queue with first-in first-out (FIFO) semantics for each active class. Nevertheless, the UML CS profile supports multiple different queue disciplines where queues are being used. For example, a port can have an input queue for incoming signals. The default setting is that all ports share a common input queue for the process they are associated. The default queue discipline is first-in first-out (FIFO) strategy. This is compatible with the process input queue model of SDL-2000.

However, the UML CS is not bound to this predefined queuing strategy. UML CS is intended to give extensibility and customizability to formal system descriptions. For example, the formal description technique ESTELLE supports a common input queue at the interaction points (port instances) as well as an individual queue for each port [Hog89]. The vision is that additional mapping implementations can map UML CS models to ESTELLE based descriptions of systems by means of the same models that can be mapped to SDL.

To enable multiple queuing strategies, a Scheduler interface is defined. A Scheduler can be assigned to ports and agent within the UML CS model. The modeling tool or mapping implementation shall also be aware of the specified Scheduler or shall be able to compile the Scheduler implementation (e.g. also specified by means of a UML CS model). For instance, a mapping implementation which is only able to map a UML CS model to an SDL specification should report an error if the scheduler differs from the SDL semantics – or it should be able to replicate the same semantics by means of various mechanism (e.g. creating port-emulating processes as a replacement for ports). This interface specifies the operations which shall be supplied by a Scheduler implementation.

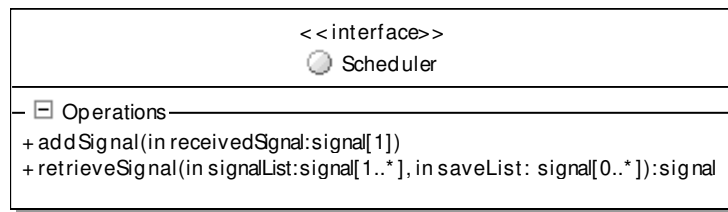


Figure 62: Interface Definition of Scheduler

As depicted in Figure 62, a Scheduler is an interface. If a queue discipline will be defined, it must implement this Scheduler interface. The Scheduler defines two operations: *addSignal* and *retrieveSignal*. The detailed BehavioralFeature of the two operations are:

addSignal(in *receivedSignal*: *signal*[1]{*is ordered=false; is unique=false*}):

addSignal adds a received signal to its internal data structure (possibly a queue or pool). It is not restricted in any way how this is accomplished or maintained. It is also not specified if the input pool or queue has an infinite size or if it is limited. If it is limited, the replacement strategy is up to the concrete implementation of a Scheduler.

retrieveSignal(in *signalList*: *signal*[1..*]{*is ordered=false; is unique=false*}, in *saveList*: *signal*[0..*]{*is ordered=false; is unique=false*}, return *selectedSignal*: *signal* [0..1]{*is ordered=false; is unique=false*})

retrieveSignal retrieves a single signal from its internal signal queuing structure. The signal is removed then. The first parameter *signalList* specifies a list of signal candidates which are allowed to be retrieved. An optional list of signals, the *saveList*, may be specified. This is a list of signal which are specified to be saved during this retrieval process until the next signal query is due. The result value *selectedSignal* returns the signal selected by the internal queue strategy. It is also possible to return an empty list. This means that a signal specified by the *signalList* has not been received so far.

Two pre-defined Schedulers are currently available and act as default values for a process and for a port: For processes, the *sdlScheduler* is assigned as the default Scheduler which has the same queue discipline as described in the dynamic semantics of SDL-2000. There is no Scheduler assigned to ports by default. That implies that all elements received are simply forwarded. This is intended for ports to have no event queue as in SDL-2000. As noted, if the schedulers are set to a different scheduler than the default assigned schedulers, this may impose different mapping techniques to SDL. However, the remainder of this thesis assumes that SDL-2000 input queue semantics are used.

7.1.3 Name resolution

Names used within a UML system specification shall be re-solved according to the UML name binding rules. However, the naming rules of UML are very loose and cannot be used for the naming used in other formal description techniques. The following stereotype constraints the available characters for names with respect to the SDL-2000 grammar for names.

NamedElement is extended by «namedElement» from the metaclass NAMEDELEMENT (FROM KERNEL, DEPENDENCIES). A «namedElement» has the multiplicity one and is therefore a *required* stereotype. This stereotype serves the purpose that a coherent mapping to other formal description techniques is possible.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
NamedElement	- no specific notation -	7.3.33 NamedElement
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«namedElement»	NamedElement (from Kernel)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context NamedElement inv: self.name->notEmpty() inv: isConformant(self.name) namedElement::isConformant(n: String, i: Integer): Boolean let ch: String = n.substring(i, i+1) in if ch->size()=0 then result=true else if letter->exist(ch) then result=isConformant(n,i+1) else result=false endif letter: Set{String} = {"a","b","c","d","e","f","g","h","i","j", "k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z", "A","B","C","D","E","F","G","H","I","J","K","L","M","N", "O","P","Q","R","S","T","U","V","W","X","Y","Z", "0","1","2","3","4","5","6","7","8","9","_"}		A <u>name</u> must only consist of uppercase or lowercase Latin letters, decimal digits or underscore '_'. A <u>name</u> must not be empty and must contain at least a letter or the underscore '_'. This stereotype is a <i>required</i> extension.

7.2 Structure

The static structure or architecture of a specification gives information on which entities take part in the description and which entities are communicating with each other. It further allows logical and hierarchical (de-)composition of closely or loosely coupled entities. This section describes the structural mapping by defining stereotyped extension to the UML metaclasses. This also enables a mapping to SDL.

In this section, the following UML metaclasses are extended by stereotypes: Package, Class, Operation, Interface, Signal, Generalization, Port, Connector, InformationFlow and InformationItem.

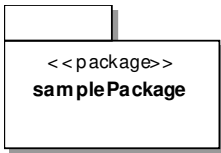
The main structuring element of a definition is an agent. An agent is either a system, block or process whereas a system is a special block. An agent may contain other agents except for a process that may contain only processes. A block executes its embedded state machines concurrently, whereas the embedded state machines of a process are executed using interleaving on transition base.

CLASS

The following graphical elements mainly apply to the specification of class (diagrams). All agents, which are a kind of system, block or process, are stereotyped extensions from the metaclass Class.

7.2.1 Package

A package is extended by «package» from the metaclass PACKAGE (FROM KERNEL). A «package» gives a common namespace to all its contained definitions.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Package		7.3.37 Package
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«package»	Package (from Kernel)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Package inv: self.name->notEmpty() inv: self.visibility=VisibilityKind::public inv: self.packageMerge->isEmpty()		Package merging is not supported.


Semantics

A «package» gives a common namespace to all its containing agents and definitions. This allows a logical grouping of the agents and of all operations, signals, attributes being defined. Therefore, a package enables re-using of previous specifications. Furthermore, a package restricts the visibility of attribute and operations to agents that are specified outside the package (see the *package* visibility modifier for attributes and operations).

The ownedMember attribute specifies the contained types of agents, variables and signals. The nestedPackage defines further contained packages.

7.2.2 System

The System is extended by «system» from the metaclass CLASS (FROM COMMUNICATIONS). A «system» specifies the outermost block of the specification. All specifications are only allowed within a «system» with the the exception of packages which must not be part of a system definition.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Class		13.3.8 Class
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«system»	Class (from Communications)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Class		A system is not an active class and therefore

<pre> inv: self.isActive=false inv: self.name->notEmpty() inv: self.classifierBehavior->isEmpty() inv: self.nestedClassifier-> forAll(c isStereotypedBy(c,block)) or isStereotypedBy(c,process)) inv: self.allInstances()->size()=1 inv: self.superClass()->size()<=1 </pre>	<p>specifies no behavior. There must be a <u>name</u> assigned. A system must only contain processes or blocks. There is only one system instance allowed.</p>
--	--


Semantics

A «system» is the UML CS representation or description of a system. A system defines the outermost block of the specification. A system type of a class is a singleton class. That is, exactly one system must be defined if no package is defined. The system separates the environment from its contained set of agents. Signals can be conveyed to the border of the system through the system's port associations and the defined interfaces.

The nestedClassifier attribute consisting of classes extended by block or process specifies all contained agents within the system. Remaining elements in the nestedClassifier attribute are part of this system and define – depending on their stereotype – a data type, an interface or a signal. ownedAttribute defines the system-visible variables of this system. The ownedConnector and ownedPort define the channels and ports at system level. ownedOperation defines system visible operations. If superClass is not empty, the system inherits its properties from a parent system (see Section 7.2.8 for details on generalization). Use of packages is defined using PackageImport notation.

7.2.3 Block

The Block is extended by «block» from the metaclass CLASS (FROM COMMUNICATIONS). A block is an agent to allow logical separation or composition of processes. Blocks are not required to have a direct physical correspondence on the target system. Blocks provide notational means to allow hierarchical structuring and to retain clearness and manageability of the specification.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Class		13.3.8 Class
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«block»	Class (from Communications)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
<pre> context Class inv: self.isActive=false inv: self.name->notEmpty() inv: self.classifierBehavior->isEmpty() inv: self.superClass->size()<=1 inv: self.nestedClassifier-> </pre>		A block agent may contain blocks and/or processes but no system.

forAll(c isStereotypedBy(c,block) or isStereotypedBy(c, process))	
---	--

Semantics

A «block» gives a logical structure to its contained agents. Communication with its containing parent is enabled by communicating with its environment. The environment as an agent within this block conveys signals to its parent block (or system) or inside this block. All agents nested in this block are executed concurrently. A block is an explicit definition of an agent type. Instantiation of this type are deployed within a composite structure.

Variables and operations declared in a block have its scope according to its visibility specified:

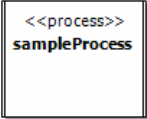
- *Public (+)*: the attribute or operation is visible to all other agents.
- *Private (-)*: the attribute or operation is visible only to its owning agent.
- *Protected (#)*: the attribute or operation is visible only to its owning agent and to all agents being a specialization of the owning agent.
- *Package (~)*: the attribute or operation is visible only to agents defined in the same package.

The term *visibility* includes the fact that neither the variable nor the operation can be read, written, accessed, referenced or called by an agent that is not included in the visibility scope.

The nestedClassifier attribute consisting of classes extended by block or process specifies all contained agents within the block. Remaining elements in the nestedClassifier attribute are part of this block and define – depending on their stereotype – a data type, an interface or a signal. ownedAttribute defines the block-visible variables of this system. The ownedConnector and ownedPort define the channels and ports at block level. ownedOperation defines block-visible operations. If superClass is not empty, the block inherits its properties from a parent block (see Section 7.2.8 for details on generalization). Single generalization of block agents is supported. That is, a specialized block agent can add ports, processes and new blocks. Re-definition (overwriting) of agents is not allowed. Specialization of block agents must follow the Liskov-substitution principle (described in more detail in Section 7.2.8). Use of packages is defined using PackageImport notation.

7.2.4 Process

The Process is extended by «process» from the metaclass CLASS (FROM COMMUNICATIONS). A process defines behavior by specification of an enhanced communicating finite state machine.

UML NODE TYPE	UML NOTATION	REFERENCE
Class		13.3.8 Class
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«process»	Class (from Communications)	self: Pid sender: Pid offspring: Pid parent: Pid

<i>OCL CONSTRAINTS</i>	<i>INFORMAL CONSTRAINTS</i>
context Class inv: self.isActive=true inv: self.name->notEmpty() inv: isStereotypedBy(self.classifierBehavior, stateMachine) inv: self.superClass->size()<=1 inv: self.nestedClassifier-> forAll(c isStereotypedBy(c,process))	A process agent must only contain other processes if any. Multiple inheritance is not allowed. A process agent must have a <u>classifierBehavior</u> of type stateMachine.

Semantics

A «process» is an active class that communicates with its environment by means of signals. The communication is done through specified interaction points, namely ports. A process owns an observable behavior. The definition of a process is done by the specification of an enhanced finite state machine. That is, the state machine defined in a process may own variables and assigns values to them. The state machine also has the ability to receive signals and to send signals to other agents which belong to its environment. This enables inter-process communication between the instances of the distributed system. If a process agent contains another process, an interleaved concurrency scheme for execution of the processes is applied.

Different to passive classes (for instance, signals), a process may also react to the reception of signals if it is specified to do so. Passive classes are only allowed to react on a signal reception with a behavior. This may introduce effects such as race-conditions. Variables and operations declared in a process have their scope according to their specified visibility:

- *Public (+)*: the attribute or operation is visible to all other agents.
- *Private (-)*: the attribute or operation is visible only to its owning agent.
- *Protected (#)*: the attribute or operation is visible only to its owning agent and to all agents being a specialization of the owning agent.
- *Package (~)*: the attribute or operation is visible only to agents defined in the same package.

The term *visibility* includes the fact that neither the variable nor the operation can be read, written, accessed, referenced or called by an agent that is not included in the visibility scope.

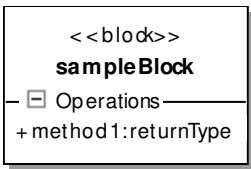
The self tag definition is the process identification value of type *Pid* assigned by the executing (virtual) machine. This value is unique within the system and is read-only. This value unambiguously specifies one process within the system and does not change during the lifetime of the execution run of the system. However, it is not guaranteed that a process gets the same process identification assigned in each run of the system. The concrete value is lost after termination of the system. The sender contains the *Pid* of the process from which a signal has been received most recently. It is undefined if no signal has been received. offspring holds the *Pid* of a created agent by this agent. parent contains the *Pid* of the agent that has created this agent. All values are read-only.

The nestedClassifier attribute consisting of classes extended by process specifies all contained process agents within the process. Remaining elements in the nestedClassifier attribute are part of this process and define – depending on their stereotype – a data type, interface or signal. ownedAttribute defines at least (depending on the visibility modifier) process-visible variables of this system. The ownedConnector and ownedPort define the channels and ports at process level. ownedOperation

defines process operations. If `superClass` is not empty, the process inherits its properties from a parent process (see Section 7.2.8 for details on generalization). Single generalization of process agents is supported. That is, a specialized process agent can add ports and processes. Re-definition (overwriting) of agents is not allowed. Specialization of process agents must follow the Liskov-substitution principle (described in more detail in Section 7.2.8). Use of packages is defined using `PackageImport` notation.

7.2.5 Operation

An Operation is extended by `«operation»` from the metaclass `OPERATION` (FROM COMMUNICATIONS). A method defines its behavior by means of a state machine or activity. The default visibility of an operation is *protected*. This implies that the operation is only visible to the agent where it is declared and to its owned agents.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Operation		13.3.21 Operation
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
<code>«operation»</code>	Operation (from Communications)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Operation inv: self.precondition->isEmpty() inv: self.bodycondition->isEmpty() inv: self.postcondition->isEmpty() inv: self.behavior-> oclIsKindOf(stateMachine) or self.behavior->oclIsKindOf(activity) inv: isStereotypedBy(self.class, process) or isStereotypedBy(self.class, block) or isStereotypedBy(self.class, system) or isStereotypedBy(self.class, package) inv: self.raisedException->isEmpty()		An operation must not have any pre-, body-, or postconditions constraints applied. The behavior must be specified by either a state machine or an activity. The owner of an operation can only be a package or agent. Exceptions shall not be defined.

Semantics

An `«operation»` owns a parameterizable behavior that is its method. The method's behavior is specified by a state machine or an activity. An operation can be invoked through the activity `CallOperationAction`, as described in Section 7.4.6. Operation invocation is only possible if the visibility rules apply and the operation is within the scope of the calling process. The operation invocation is done synchronously. That is, the calling process is suspended until the invoked method

has completed its activity. After termination of the invoked behavior, the return value is passed back to the calling process which is then resumed.

Variables and operations declared in an agent have their scope according to their specified visibility:

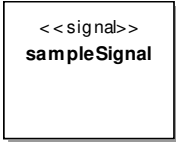
- *Public (+)*: the attribute or operation is visible to all other agents.
- *Private (-)*: the attribute or operation is visible only to its owning agent.
- *Protected (#)*: the attribute or operation is visible only to its owning agent and to all agents that are a specialization of the owning agent.
- *Package (~)*: the attribute or operation is visible only to agents that are defined in the same package.

The term *visibility* includes the fact that neither the variable nor the operation can be read, written, accessed, referenced or called by an agent that is not included in the visibility scope.

The ownedParameter attribute defines the formal parameters of the operation and its associated method. The method defines the behavior of this operation and shall only associate a state machine or an activity.

7.2.6 Signal

A signal is extended by «signal» from the metaclass SIGNAL (FROM COMMUNICATIONS). A «signal» is a specification of a type of information for send request instances communicated between processes. The receiving process handles the signal instance according to its specified behavior. Data values can be carried by a send request and passed onto it. The send invocation event that has caused the request can convey information by the signal which is represented as attributes of the signal instance. Signals are defined independently of the processes handling the signal, but within the visibility scope.

UML NODE TYPE	UML NOTATION	REFERENCE
Signal		13.3.23 Signal
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«signal»	Signal (from Communications)	priority: Integer = 0 sender: Pid
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context Signal inv: self.general->size()<=1 inv: self.extension_signal.priority>=0 and self.extension_signal.priority<256 inv: self.ownedOperation->isEmpty()		The optional <i>priority</i> attribute defines a possible precedence of this signal. A higher value specifies a higher priority. The tagged value <i>sender</i> represents the sender process identification that has executed the output statement sending this signal.

Semantics

A «signal» triggers a behavior in the receiver in an asynchronous way. That is, the sender does not get a response to its sent signal. The sender of a signal will not block waiting for a response but continues its execution immediately. A process specifies that its instances will be able to receive that signal by declaring a reception associated to a given signal. The process will respond to it with the designated behavior.

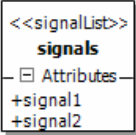
The priority can be specified with a value to a signal. However, the semantics of the priority value is up to the port scheduling algorithm that is being used. With SDL semantics by default, this value is ignored by the process where the signal has been received. The concrete range of possible priority values is chosen independently from any existing priority rules.

The sender is the process identification of the process that has sent the signal by invoking an output action (see Section 7.4.5). This value is implicitly set by the executing machine when the output is executed. A signal can be specialized and further attributes can be added.

The ownedAttribute attribute defines the attributes of this signal. The general attribute defines the generalized signal.

7.2.7 SignalList

A signal is extended by «signalList» from the metaclass SIGNAL (FROM COMMUNICATIONS). A «signalList» is a defined composition of several signals.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Signal		13.3.23 Signal
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	<i>TAGGED VALUES</i>
«signalList»	Signal (from Communications)	ownedSignal: Signal [0..*]
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Signal inv: self.ownedOperation->isEmpty() inv: self.ownedAttribute->isEmpty() inv: self.general->size()=0 context signal inv: self.ownedSignal->asSet()=ownedSignal		A signalList shall only contain other signals. Generalization is not supported. ownedSignal shall not contain duplicates.


Semantics

A «signalList» specifies a list of several signals and groups them under a single name. It can be used as shorthand to define multiple signals on a channel. A «signalList» cannot be used in a SignalEvent or Output.

The ownedSignal tag definition specifies the set of signals it represents.

7.2.8 Generalization

A generalization is extended by «generalization» from the metaclass GENERALIZATION (FROM KERNEL, POWERTYPES). A «generalization» implements object-orientated features to the profile. Specializing a process allows re-definition of states and adds transitions and actions to the derived process. A specialization can be applied to processes, blocks, systems, ports and primitives.

UML NODE TYPE	UML NOTATION	REFERENCE
Generalization		7.3.20 Generalization
UML CS STEREO TYPE	UML META CLASS	
«generalization»	Generalization (from Kernel, PowerTypes)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context Generalization inv: self.generalizationSet->size()=0 inv: self.specific.parent()->size()=1 inv: (self.general.oclIsTypeOf(process) and self.specific.oclIsTypeOf(process)) or (self.general.oclIsTypeOf(block) and self.specific.oclIsTypeOf(block))		Multiple inheritance is not supported on any type. Specialization is only possible between same types.

Semantics

For specialization, the Liskov substitution principle applies which provides a guideline to sub-typing any existing type [LW93]:

Let $q(x)$ be a property provable about objects x of type T . Then $q(y)$ should be true for objects y of type S where S is a subtype of T .

In other words, all operations with a reference to the base class should be completely transparent to the type of the inherited object. It should be possible to substitute an object of one type with another within the same class hierarchy. Inheriting classes must not perform any actions that will invalidate the assumptions made by the base class.

Let M be a state machine defined by process with $M = (S, T)$ and let M' be a process generalized by M with a state machine $M' = (S', T')$. Let (S'', T'') be a set of states and transitions that specialize M , the following rules must apply:

$$S' = S \cup S''$$

$$T' = T \cup T''$$

These rules imply that for a specialization of a process, states and transitions cannot be removed. It is only allowed to add states and transitions to the specialized state machine. The observable behavior of a class to its generalized class must not differ.

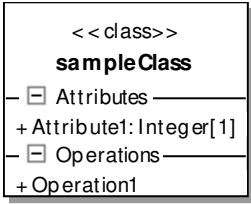
When an operation is called which may be specialized from a base class *object-oriented resolution* applies. That is, when an operation call request is received, the class of the target object is examined whether it defines an operation with matching signature (its matching formal parameters). If such a matching operation is found, the behavior associated as method is the result, thus completing the

resolution. If not, the general (or parent) classifier is examined for a matching operation and so on up the generalization hierarchy. This resolution procedure is repeated until a method is found or the base agent of the system specification is reached. As this profile does not support multiple parent classes (multiple inheritance), there is only one single generalization look-up path to be examined. If a method is found in one of the ancestor classes, then this method is the result of the resolution process.

The general attribute defines the generalized agent; the specific attribute specifies the specialized agent.

7.2.9 Class

A Passive Class is extended by «class» from the metaclass CLASS (FROM COMMUNICATIONS). A «class» defines a class for object instances that have no own thread of control and do not perform any activity upon instantiation. Invocation of a behavior can only be triggered from other active agents. Instances of this class definition can be created by using CreateObjectAction (see Section 7.4.4). A passive class merely acts as a cohesive container for several operations for remote procedure calls.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Class	 <pre> classDiagram class sampleClass { +Attribute1: Integer[1] +Operation1 } </pre>	13.3.8 Class
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«class»	Class (from Communications)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Class inv: self.isActive=false inv: self.name->notEmpty() inv: self.classifierBehavior->isEmpty() inv: self.nestedClassifier->forAll(c isStereotyped(c,class))		A class shall not define any <u>classifierBehavior</u> . Behavior can only be performed by invoking operations. A class shall only define nested passive classes, but no active classes.

Semantics

A passive «class» describes a collection of encapsulated instance variables and operations (methods) together with the implementation of those types. It is a cohesive package consisting of a particular kind of compile-time metadata for its instances. This is very similar to an object data type described in Section 7.7. However, different from an object type this class must not define infix operations.

ownedBehavior defines the activities or state machine which are invoked by the operations. ownedAttribute defines the attributes according to a block (see Section 7.2.3).

The class describes the properties and rules by which objects behave. These objects are referred to as *instances* of that class. A class specifies the structure of data that each instance contains. It further defines the operations that can access and modify this data and perform actions. A method is the

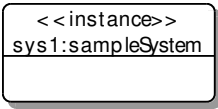
implemented function of an operation with a special property having access to data stored in an object. A class is the most specific type of an object in relation to a specific layer. Instances of a class will have certain features, attributes or properties in common. A behavior of a passive class is executed immediately with call event. This is different to agents, where a triggered behavior is only executed if the agent's behavior enables this. It can be observed that this might raise race-conditions when multiple concurrently executed methods access the same properties of a passive class or its instances.

COMPOSITE STRUCTURE

The following stereotypes apply to elements specified in a Composite Structure. Instances from agent types are created as parts of a StructuredClassifier. The StructuredClassifier is at least a system agent. It is not explicitly needed to be specified. A StructuredClassifier has not to be shown in a diagram. It can also be omitted in case the modeling tool lacks support for it and the association from the agent to this composite structure is still clear.

7.2.10 Instance

A Class is extended by «instance» from the metaclass CLASS (FROM STRUCTUREDCLASSES). An instance is used to specify the nested agents within an agent and the communication paths by means of channels and ports. This stereotype is only provided for UML tools that provide no direct access to the model repository.

UML NODE TYPE	UML NOTATION	REFERENCE
Class		9.3.1 Class
UML CS STEREOTYPE	UML METACLASS	
«instance»	Class (from StructuredClasses)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context Class inv: self.name->notEmpty()		The <u>name</u> of the instance shall not be empty.

Semantics

An «instance» is the concrete agent instantiation within a composite structure. The parts of the instance decompose into the compositions of the agent, e.g. blocks or processes. The execution semantics is a concurrent scheme for processes and blocks defined within a block instance. Processes of process agents are executed in an interleaved scheme. An instance does not need to have a name and can also be anonymously instantiated by using the following expression:

```
«instance» ::= [<agent name>] ':' <agent type identifier>
```

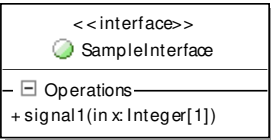
or, as it is clear from the syntax above, for anonymously defined instances the following expression:

```
'::' <agent type identifier>
```

The explicit instantiation is done by the first expression with a distinct name assigned to the agent where the second expression instantiates the agent anonymously with an unknown name.

7.2.11 Interface

An interface is extended by «interface» from the metaclass INTERFACE (FROM COMMUNICATIONS).

UML NODE TYPE	UML NOTATION	REFERENCE
Interface		13.3.15 Interface
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«interface»	Interface (from Communication)	ownedSignal: Signal[0..*]
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context Interface inv: self.ownedReception->size()>0 implies extension_interface.ownedSignal= self.ownedReception->collect(signal) inv: self.redefinedInterface->notEmpty() implies self.ownedAttribute->includesAll(self. redefinedInterface.ownedAttribute) and self.ownedOperation->includesAll(self. redefinedInterface.ownedOperation) and self.ownedSignal->includesAll(self. redefinedInterface.ownedSignal)		Signals have to be associated by means of the Reception metaclass. <u>ownedSignal</u> is derived if <u>ownedReception</u> association is not empty. Otherwise, <u>ownedSignal</u> specifies a collection of signals.

Semantics

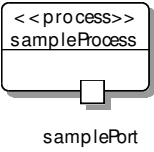
An «interface» is the declaration of the ability of an agent that these signals and operations are expected and can be handled. An interface can mainly be interpreted as a kind of an agreement that formalizes that the specified signals can be sent and received to convey information. It does not provide any details how the requested behavior is performed.

Therefore, an interface is used to give a concrete specification of the interaction point of an agent, called a *port*. Such a port defines the signals and operations that it can convey to and from its environment. The signals of this interface have to be associated by means of the *Reception* metaclass which specifies the signals a classifier is willing to accept.

As shorthand, the ownedSignal tag definition allows to specify a list of signals defining the signals that become a part of the interface's contract. This shall only be used if the ownedReception association is empty. ownedSignal or ownedReception defines the list of signals, ownedAttribute the attributes and ownedOperation the operations the binding agent requires or provides.

7.2.12 Port

A port is extended by «port» from the metaclass PORT (FROM PORTS). A port defines the kind of information an agent is able to receive and what the agent is able to send to other agent instances.

UML NODE TYPE	UML NOTATION	REFERENCE
Port		9.3.11 Port
UML CS STEREO TYPE	UML METACLASS	TAGGED VALUES
«port»	Port (from Ports)	queueDiscipline: Scheduler[0..1] isDynamic: Boolean = false <u>instances: Integer</u>
OCL CONSTRAINTS	INFORMAL CONSTRAINTS	
context Port inv: self.isBehavior=self.class-> oclIsTypeOf(process) inv: self.isService=true inv: self.type->oclIsKindOf(interface) inv: self.redefinedPort->notEmpty() implies self.name=self.redefinedPort.name inv: self.aggregationKind= AggregationKind::composite inv: self.extension_port.isDynamic implies self.extension_port.instances>0 inv: not self.isDynamic implies self.upper=1	The optional <u>queueDiscipline</u> specifies if inbound signals are queued by a specific discipline by this port until they are consumed by a process. A port must be type of the interface it provides. Ports are behavior ports for process agents as the classifier's behavior may be invoked upon signal events.	

Semantics

A «port» is the definition of an interaction point of an agent with its environment. The required and provided interfaces of the port specify the signals or operation calls that are allowed to be conveyed through this port. A port encapsulates the agent from its environment. That is, the agent implementation can be changed without any further adaptations as long as the provided and required interfaces remain unmodified and realized.

The requiredInterface attribute defines the signals and operations that are expected to be received by the agent's environment. The providedInterface defines the signals and operations that are expected by the associated agent. A port may re-define a port when its associated block or process is specialized. This allows adding interfaces to the existing definition of the required or provided interfaces. A port may have a scheduler for the received signals which is specified in the queueDiscipline tag definition. This implements different queue disciplines for the receive buffer. The concept of multiple queuing strategies and a Scheduler definition is explained in Section 7.1.2.

Dynamic Ports

Ports can be declared as being dynamic if its tag definition isDynamic is set to true. That means, multiple ports can become property of an agent but are not required to establish a connection by means

of a channel. Several dynamic ports establish a *dynamic port group*. All members of one *dynamic port group* are identified by the same port name. If a modeling tool does not allow multiple identical port names, the name should be suffixed by a sequence of numbers in brackets (e.g. if the ports' names would be *samplePort*, then an alternative name would be *samplePort(0)*, *samplePort(1)*, ...).

Dynamic ports are allowed to stay un-connected on agent instances. That is, no channel is attached to them. During initialization of the system, this port is implicitly removed. This may also apply to a channel path between two dynamic ports. Following Figure 63 illustrates this property:

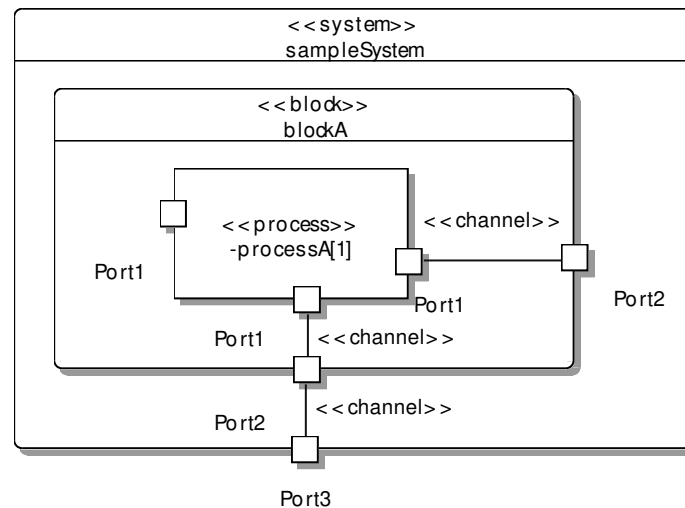


Figure 63: Dynamic Ports and Channel Paths

In this *sampleSystem*, the process agent *processA* has three dynamic ports attached. Block agent *blockA* has two dynamic ports attached named *Port2*. If *Port3* of the system is also a dynamic port, it does not have any impact on the following procedure. During mapping, the unconnected *Port1* on the left is implicitly removed. This also applies to *Port1* and *Port2* including the channel in-between. This is the result that *Port2* is connected neither within nor outside of its encapsulating agent. The dynamic ports *Port1* and *Port2* on the bottom are not removed because they have an un-interrupted channel route between two processes or systems.

Dynamic port classes are defined without specifying their amount. The amount of ports within a dynamic port group is derived from the final model. It is saved to the attribute instances by the execution environment. instances is always set to 1 for a simple port, but may be higher for dynamic ports. The concept and rationale for the need of dynamic ports is described in Section 5.2. Note that it is required to have identical names for all ports of the dynamic port group as the system has to be aware that a variable amount of ports may be present from run to run. It would not be possible to enumerate all dynamic ports of a group if all members of a dynamic port group had a different name. There is no way to inform the system about this port configuration change when the port names are hard-coded within the system description. A dynamic port with multiple instances associated with the same agent features a dynamic port group. They all have the same name identifier. A distinct dynamic port can be addressed by the following expression

```
<dynamicport identifier> ::= <port name> '(' <port instance> ')'  
<port name> ::= <identifier>  
<port instance> ::= <natural literal name>
```

Note that the first port instance is a dynamic port addressed by a zero index. This implies that the instance index must not be equal or greater than the instances tag definition of the dynamic port class. For example, let the textual notation for a signal input via a dynamic port *dynport* be as follows

```
INPUT signal1 via dynport(0)
```

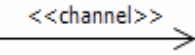
Analogously, let the notation for a signal output via a dynamic port *dynport* be as follows:

```
OUTPUT signal1 via dynport(0)
```

It is not specified which port is mapped to which instance when a system starts execution. Therefore, an agent cannot rely on the fact that the mapping from a dynamic port index to the instantiated ports will always map to the same concrete port instance. This has to be assured by the corresponding agent itself, e.g. by means of generating a mapping between a dynamic port index and the process identification when signals have been received through this port.

7.2.13 Channel

A channel is extended by «channel» from the metaclass CONNECTOR (FROM INTERNALSTRUCTURES). A channel explicitly connects two port instances together.

UML NODE TYPE	UML NOTATION	REFERENCE
Connector		9.3.6 Connector
UML CS STEREO TYPE	UML METACLASS	TAGGED VALUES
«channel»	Connector (from InternalStructures)	signalList0: Signal[0..*] signalList1: Signal[0..*] delay: Boolean = false distinctSignals: Boolean = false flow: InformationFlow[0..*]
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
<p>context Connector</p> <p>inv: self.end->size()=2</p> <p>inv: self.type->isEmpty()</p> <p>inv: self.end[0]<>self.end[1]</p> <p>inv: self.end->select(c c.partwithPort->isEmpty()->isEmpty()->isEmpty()</p> <p>inv: self.redefinedConnector->isEmpty()</p> <p>context channel</p> <p>inv: self.signalList0->notEmpty() or self.signalList1->notEmpty()</p> <p>inv: self.flow->size()>0 implies signalList0 = self.flow->select(f f.target= self.base_Connector.end[0].role)->collect(i i.conveyed)->select(j j.represented->oclIsKindOf(Signal))</p> <p>inv: self.flow->size()>0 implies signalList1 = self.flow->select(f </p>		<p>A «channel» is either unidirectional or bidirectional. A channel must not connect a port with the same port. <u>name</u> may be empty as channel-specific addressing is not allowed. The set <u>signalList0</u> or the set <u>signalList1</u> have to contain at least one element (otherwise, this channel would not be able to convey any signal in any direction)</p> <p>A channel cannot be redefined.</p> <p>The target port shall specify the same signals in its <i>provided</i> interface.</p> <p><u>signalList0</u> and <u>signalList1</u> are derived values from the appropriate InformationFlow (see 7.2.14) interface and signal specification.</p> <p>If a channel's <u>connectorEnd</u> defines higher multiplicity than one, its multiplicity value shall be equal to the multiplicity of the</p>

<pre>f.target=self.base_Connector.end[1].role)-> collect(i i.conveyed)->select(j j.represented->oclIsKindOf(Signal))</pre>	connected agent (already constrained by metaclass).
---	---

Semantics

A «channel» establishes a unidirectional or bidirectional communication path between two port instances. This implies that a channel connects two agents to allow the conveyance of signals between them. A channel shall not have a name. That is caused by the fact that the explicit output addressing of signals is only implicitly possible (a port is selected which allows the conveyance of that signal by its requiredInterface) or by the explicit specification of an outbound port. However, a channel cannot be addressed. If the signal has no *via*-port specified, its outbound port is specified by the requiredInterface association.

The channel defines the tags signalList0 and signalList1 with a type of set of signals. Both sets specify the signals that are allowed to convey to the target port. If one of the sets is empty, the channel is unidirectional. In this case, a modeling tool should visualize this by showing a directed arrow on the channel. Both sets must not be empty at the same time. The signalList0 specifies the signal that are allowed to convey to the end[0] port specification. The signalList1 specifies the signals that are allowed to convey to the end[1] port specification. Both attributes are an un-ordered set of signals. Only a signal being an element of this set can be sent by a process through this channel. Both signalLists are derived values from the appropriate InformationItems that specify the communication objects on this channel. If InformationItems are not available in a modeling tool, the signalList tag definitions can be used.

The delay tag definition specifies if the channel is a delaying channel. Signals conveyed through delaying channels are delivered to the recipient within a specific time. The concrete amount of time delivery remains un-specified within the profile and it is up to the specific mapping to a FDT. If the delay is not true, the conveyance of signals is assumed to be done in zero-time.

When a signal is to be sent through a specific port and there are two channels connected to this port that are able to convey this signal, an arbitrary one of both channels is selected. This can be avoided by the distinctSignals attribute. The distinctSignals attribute specifies whether this channel allows to be arbitrarily chosen. If this attribute is set to a true value, there shall be no other channel connected to the same port that also allows conveyance of at least one identical signal or operation call. The modeling tool has to verify whether the model is ill-formed by these coinciding channels. If distinctSignals is set to false – this is the default – multiple channels are allowed which also convey the same signal type. Enabling this, prevents non-determinism and arbitrary channel selection for (a)synchronous signals at the connected ports. The channel selection procedure is described in the following section.

7.2.13.1 PORT AND CHANNEL SELECTION

When a signal is to be sent by an activity (see Section 7.4.5), it is possible to implicitly or explicitly specify the port through which the signal is to be conveyed. Explicit specification addresses the port unambiguously as there must be a unique port name within each agent. In its requiredInterface association the port instance must specify that it is able to convey this signal. If this does not match, the model is considered ill-formed. A matching channel has to be connected to this port with its inbound end. That is, each signal definition in the signal list of the requiredInterface specification has to match the same signal definition in at least one channel being connected to this port.

If the port confirms to convey the signal, a channel must be connected to that port. If the channel is connect via its end[0] attribute, the signalList1 set specifies which signal can be conveyed to the target

port. If the channel connected to the port via its `end[1]`, the `signalList0` specifies the set of signals. A signal can only be sent if it is an element of the corresponding `signalList` of the channel. If there is more than one channel defining this signal, an arbitrary channel is selected in a non-deterministic way.

7.2.13.2 CHANNEL MULTIPLICITY

Multiplicity can be specified on the `ConnectorEnd` association by `end` defining the amount of channels connected to a single instance. When it is not specified, the corresponding instances are connected peer-to-peer. Figure 64 shows such a single channel multiplicity.

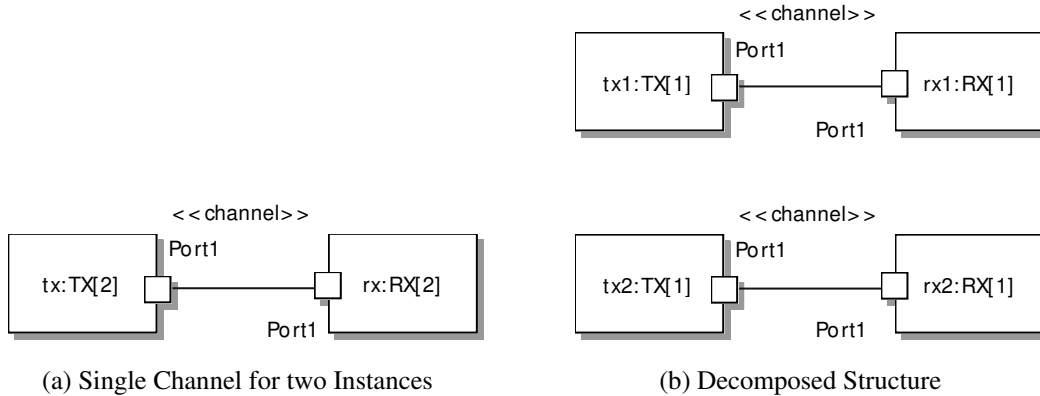


Figure 64: Decomposition of Multiple Instances with Single Channel Multiplicity

If the multiplicity is specified, the corresponding instances are connected like a star-topology, as shown in the following Figure 65. The multiplicity value of a channel's end either has to be one or has to match with the multiplicity of the connected agent instance.

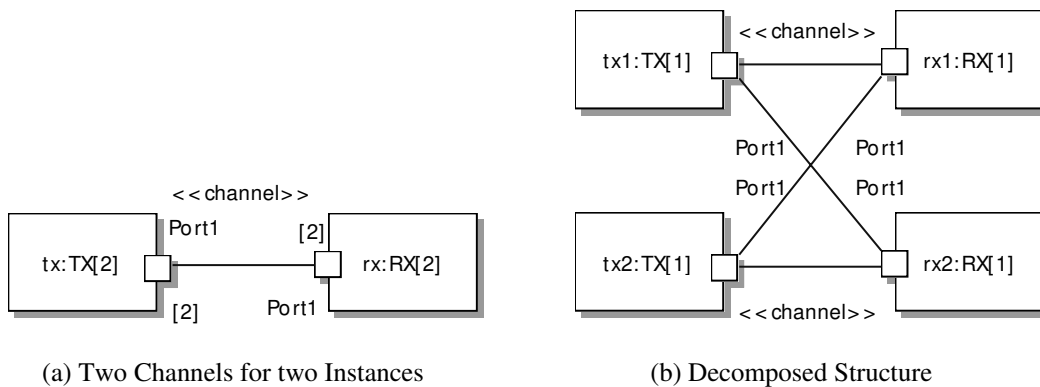


Figure 65: Decomposition of Multiple Instances with Channel Multiplicity of Two

7.2.14 InformationFlow

An `InformationFlow` is extended by `<<informationFlow>>` from the metaclass `INFORMATIONFLOW` (`FROM INFORMATIONFLOWS`). An `InformationFlow` specifies the communication between two objects and the used information objects. These information objects are the `InformationItems`.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
InformationFlow	- no specific notation -	17.2.1 InformationFlow
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	<i>TAGGED VALUES</i>
«informationFlow»	InformationFlow (from InformationFlows)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context InformationFlow inv: self.realization->isEmpty() inv: self.realizingConnector->notEmpty() inv: isStereotypedBy(realizingConnector, channel) inv: self.realizingActivityEdge->isEmpty() inv: self.realizingMessage->isEmpty() inv: self.source->forAll(p p.oclIsKindOf(Port)) inv: self.target->forAll(p p.oclIsKindOf(Port)) inv: self.conveyed->forAll(c isStereotypedBy(c,informationItem))		An informationFlow can only specify the conveyed information between ports and their connector.

Semantics

The «informationFlow» specifies a unidirectional or bidirectional communication path between the channel and the port instances. The «informationFlow» can specify «informationItems» that provide a contract of signals, operation calls or interfaces that can be conveyed through its owning channel. If the informationItem is empty, no signal can be conveyed through this channel in this direction.

The conveyed attribute specifies the extended InformationItems that can be conveyed through a channel. The realizingConnector attribute defines the channel for which it provides the signal set. The target and the source attributes define the originating and target ports.

7.2.15 InformationItem

An InformationItem is extended by «informationItem» from the metaclass INFORMATIONITEM (FROM INFORMATIONFLOWS). An InformationItem specifies the information objects between two objects.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
InformationItem	- no specific notation -	17.2.2 InformationItem
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	<i>TAGGED VALUES</i>
«informationItem»	InformationItem (from InformationFlows)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context InformationItem inv: self.represented->forAll(p		An informationItem can only be a Signal, Operation or Interface.

p.oclIsKindOf(Signal) or p.oclIsKindOf(Operation) or p.oclIsKindOf(Interface))	
--	--

Semantics

An «informationItem» represents abstract information. This can only be a signal, operation call or an interface. This «informationItem» specifies the object instances that can be conveyed through a channel.

The represented attribute specifies the items that can be conveyed through a channel defined by the corresponding InformationFlow.

7.3 Behavior

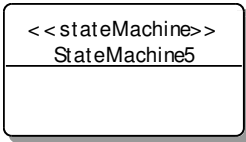
In this section, the state machine elements and the core concepts of behavior of the UML CS profile are introduced.

A *process agent* is the communicating and reactive part of a distributed *system*. A *system* consists of at least one *block* or *process* and the block contains at least one *process*. A *block* can be partitioned into one or more sub-agents for logical reasons, e.g. structuring and clarity if necessary. A block must contain another block or a process. A *process* is the active part in a system that has at least a single thread of control. The process waits for events to occur which may invoke a *trigger* and reacts to it in a certain way. This reaction is called *behavior*. A *trigger* can be invoked by an event. This may be the reception of a *signal* or condition, which yields a specific value, or even a non-deterministic event (randomness) as well. A *behavior* specifies what a trigger an agent is waiting for and in which way it reacts to it. The behavior of an agent is specified using a state machine together with activities by using the UML CS profile. An agent instance is an extended finite communicating state machine that has its own identity, its own signal input queue, its own life cycle and a reactive behavior description.

The following metaclasses of the UML are extended in this section: StateMachine, Region, State, Transition, Pseudostate and FinalState.

7.3.1 State Machine

The State Machine is extended by «stateMachine» from the metaclass STATEMACHINE (FROM BEHAVIORSTATEMACHINES). State machines are used to define the behavior focussing on reactive systems. This kind of behavior is based on the finite state machine formalism consisting of a set of states and transitions that are modeled as a traversal of a directed graph of state nodes interconnected by joined transition. Each transition is triggered by the dispatch of events and the state machine executes a series of associated activities.

UML NODE TYPE	UML NOTATION	REFERENCE
StateMachine		15.3.12 StateMachine

<i>UML CS STEREOYPE</i>	<i>UML METACLASS</i>	
«stateMachine»	StateMachine (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context StateMachine inv: self.isReentrant=false inv: self.specification->isEmpty() or self.specification->oclIsKindOf(Operation) inv: self.precondition->isEmpty() inv: self.postcondition->isEmpty() inv: self.region->size()=1 inv: self.connectionPoint->isEmpty() inv: extendedStatemachine->size()<=1 inv: self.ownedAttribute->forAll(a a.aggregation= AggregationKind::composite) inv: self.ownedBehavior->isEmpty()		A state machine shall not define any operations. Operations shall only defined within their <u>specification</u> class.

Semantics

Agents in UML CS execute an observable behavior. That is, it reacts on external stimuli, changes its internal states and can send signals. Systems and blocks also execute an observable behavior, but this is achieved by the activities of their owned processes. A state machine describes the behavior (classifierBehavior) of a process (active class) or of an operation (method). The UML 2 state machine is an informally defined graphical notation of a communicating finite state machine. It provides a state machine view with a focus on its states.

If this state machine implements the behavior of an operation, this operation shall define the specification. If the state machine implements the behavior of an agent, specification shall be empty. The nestedClassifier attribute defines data types, enumeration or primitive types. The ownedAttribute attribute defines local variables of this state machine.

This state machine has two main elements: States and transitions that connect two states. There is always exactly one distinct state where the state machine currently resides in. This is the active state. At the beginning of an execution, it is always the start state (an extension from the pseudostate of kind initial). In a process, there must be exactly one start state defined. A transition from a state sets the state machine into another state to which the transition is connected. This can occur by an event that triggers a transition originating from the active state. In UML CS, there are the following assumptions made to the state machine execution:

- The state machine is sequentially executed
- An activity can only be executed on transitions
- There is one single external event read from the input queue. The next external event will only be read from the input queue if there is no transition selectable from the active state which is enabled and triggered (*run-to-completion* principle, see below).
- Transition guards do not have any side effects.
- Events which cannot trigger any transition are discarded silently unless they are deferred (see Section 7.3.3)

A state machine partitions the processing of events into single steps. Each of them is caused by an event instance directed to the state machine. The UML CS state machine's semantics is based on the fact that each event is placed in an input queue and being processed in turn. Each event that is able to invoke a trigger is processed by a so-called run-to-completion (RTC) step. If such an RTC step is pending, it is dispatched to the state machine after the previous RTC step has been completed. Although this limits some potential communication configurations, this semantic assumption simplifies the transition function of the state machine. Any pending event is processed only after the state machine has settled into a well-defined, stable state configuration. This gives the practical meaning that a state machine does not have to cope with potential race conditions which can be invoked by multiple pending events that have to be processed concurrently. The behavior of a process is executed isolated from external events, hence providing thread protection.

Besides, it is an interesting fact that it would be possible to define the semantics of a state machine where multiple RTC steps can be applied concurrently to orthogonal regions of composite states, rather than to the complete state machine. Therefore, this would loosen the event serialization constraint. However, such semantics are subtle and hard to implement. The dynamic semantics as defined in this thesis are based on the principle to apply RTC steps to the entire state machine.

7.3.1.1 RUN-TO-COMPLETION SEMANTICS

The execution semantics of a UML state machine comprises a sequence of run-to-completion (RTC) steps. In particular, such a step is a change from one configuration (active state, variables, queues) of the state machine to another configuration. At the beginning, the sequence of steps starts in the initial configuration of the state machine with variables initialized by default or pre-defined values and empty signal queues. Then the configuration is targeted by the forward directed compound transition tree generated from the start state (the root state) of the state machine's state hierarchy. A compound transition represents the complete transition chain between two states. A compound transition can consist of several single transitions that connect the pseudostates in-between, such as decision and merge nodes.

In an RTC step of a configuration, an event is fetched from the event input queue. This triggers the selection of a maximally consistent set of enabled compound transitions – outgoing from the state of the current configuration and its guards being satisfied. If such a set – a step – exists, all its compound transitions are executed successively. In particular, the active state that is exited by the step is left by an inner to out manner and the effect of the step is executed. The state – entered by the step – is entered in an outside-in manner. In particular, when an event instance is dispatched it may result that one or multiple transitions being enabled depending on the triggers of the transitions. Especially, only transitions that are triggered by the corresponding event type can be enabled. If there is no transition enabled, the event trigger is silently discarded without any effect by default. To avoid this loss of an event trigger, a trigger can be specified as *deferred* in the active state. Hence, the trigger can be processed later. The state machine selects a subset of the enabled transition and executes them. This takes the state machine into a new active state machine configuration. This change of state machine's configuration is called a step. The procedure of selecting and executing transitions is described in detail in Section 7.3.4. While a state machine executes a set of transition, it may perform some behavior thus possibly causing additional event effects directed to itself or other agent instances. For synchronous communication, it may happen that the execution of a transition is suspended until the communicating peer completes its own run. The procedure how transitions are handled that are in conflict to each other is defined in Section 7.3.4.

7.3.1.2 COMPLETION TRANSITIONS, COMPLETION EVENTS


Completion transitions are not supported by the UML CS profile because of their possible infinite transient nature. Completion transitions are transitions that do not define a trigger although they may have a guard. A completion transition is typically taken upon the completion of the *entry*, *do* or *exit*-actions of its owning state. After having processed an event, the state machine can reach a configuration with a state having outgoing completion transitions. This is sometimes referred to as a *transient configuration*, because a subsequent configuration change is instantaneously scheduled. Such a configuration can be considered non-stable. This requires the state machine to execute further steps until it reaches a stable, non-transient configuration.

Completion transitions are triggered by corresponding completion events that are dispatched to the state machine whenever a transient configuration is encountered. As a consequence, these completion events are dispatched in a series of steps. This process of continuous dispatch ends when a stable configuration is reached. This finalizes the RTC step initiated by the event instance and control returns now to the dispatcher. A new event instance can be dispatched.

It is obvious that if this case would come to pass, a state machine may never settle down into a stable configuration. An subsequent event instance can be directed to a state machine that is suspended in the middle of an RTC step due to some other object and thus may re-trigger that transition. To overcome such cases in an implementation, the UML CS profile does not support the execution of activities during an active state configuration. Hence, a completion event cannot trigger a transition.

7.3.2 Region

The Region is extended by «region» from the metaclass REGION (FROM BEHAVIORSTATEMACHINES). A region is an orthogonal part of a state machine or of a composite state. It contains states and transitions. A region enables concurrency within a state machine specification. Concurrency within a process is not supported. There has to be only one single «region» within a «stateMachine» specification that does not imply any semantics.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Region		15.3.10 Region
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«region»	Region (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Region inv: self.stateMachine->size()=1 inv: self.stateMachine.region->size()=1		Exactly one region must be placed inside a state machine.

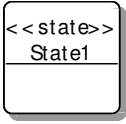
Semantics

There shall be exactly one «region» within a state machine, because concurrency within a state machine (a process) is not supported. Therefore, a «region» has no semantics.

7.3.3 State

State is extended with the stereotype «state» from the metaclass STATE (FROM BEHAVIORSTATEMACHINES).

A «state» represents a particular condition (configuration) in which the state machine of an agent waits for an event. This event may raise a trigger, for example, the consumption of a signal instance. If a signal instance is consumed, the associated transition is triggered and its specified behavior is executed. A transition may also be interpreted as the result of a condition that evaluates to true, this specific condition is called *guard*. A transition connects exactly two states or pseudostates with each other, but with one source state and one target state as both vertexes are saved in an ordered set. The next state can be considered reachable at an instant, possibly without consuming any system time. However, it is only defined that system time cannot decrease during a transition.

UML NODE TYPE	UML NOTATION	REFERENCE
State		15.3.11 State
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«state»	State (from BehaviorStateMachines)	isStateList: Boolean = false
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context State inv: self.name->notEmpty() and self.name<>'-' inv: self.doActivity->isEmpty() inv: self.entryActivity->isEmpty() inv: self.exitActivity->isEmpty() inv: self.submachine->isEmpty() inv: self.regions->isEmpty() inv: self.connectionPoints->isEmpty() inv: self.isComposite=false inv: self.isOrthogonal=false inv: self.isSimple=true inv: self.isSubmachineState=false inv: self.stateInvariant->isEmpty() inv: self.extension_state.isStateList implies self.isSimple inv: self.redefinedState->size()=1 implies self.outgoing->includesAll(self.redefinedState.outgoing)		A state does not have any activity on entry, exit or do. Behavior is only allowed to be invoked by a transition. If this state is a state list, it must not be a composite state. A state can be redefined but its transition cannot. The state name must not be equal '-'. This is reserved for history (see Section 7.3.15)

Semantics

During execution, a «state» can become active or inactive. A state becomes active when it is entered because of some transition. It becomes inactive if it is exited as a result of a transition. A state can be exited and entered caused by the same transition (e.g., self-transition). A state machine has exactly one active state in each moment during its lifetime. A state is a condition in which the state machine is

waiting for an event to occur. There is no (observable) behavior executed when entering, waiting or leaving a state in UML CS. The name attribute defines the name of the state.

The occurred event may fire a trigger that may cause a state transition of the state machine. A list of possible triggers is defined by the outbound transitions that are connected to this active state. The state machine has a signal scheduler which dispatches a signal located in its input queue to the state machine. A transition is selected on the basis of this signal scheduler and the triggers the transitions are waiting for.

Generally, a transition is active if and only if the trigger is fired and the guards of the transition are satisfied. If there is only one transition active, this transition is selected and executed. If there are multiple transitions active, the scheduler specifies which transition has a higher priority. Signals can be sent with a priority value (see Section 7.2.6). A selection scheme for a scheduler might be the following: If a transition is active and its signal has a higher priority than the signal of all other active transitions, then this transition is selected. If there is another signal in the input queue with the same priority, the one is selected which has been received first. The general rule is that for transition selection only one distinct transition can be deterministically chosen.

A state is a *soft state* if at least one of its outgoing transitions associates a TimeEvent with the transition's trigger. Hidden from the user, the modeling tool has to define a timer for each of the outgoing transition with a TimeEvent. All timers are initialized with their corresponding timeout values (according to the ValueSpecification of the TimeExpression) when the state is (re-)entered. All TimeEvent-triggers are implicitly replaced by SignalEvents which trigger the transition after the reception of a timeout signal.

7.3.3.1 STATE LISTS

A state list is an optional shorthand for a cumulative notation of states that have the same outgoing transition in common (e.g. a transition that is triggered by the reception of an abort signal). A state list is indicated by the Boolean tag definition isStateList. The proposed notation for a state list consisting of the *State1* and *State2* is *State1, State2* as defined by the following expression for a state list:

```
<state list> ::= <state name> [',' <state name>*]
```

In addition, a state list shall be specified using the notation

```
<state list> ::= '*' ['(' <state name> [',' <state name>*] ')']
```

This specifies that all states being defined within the current state machine have the transition in common. This can be extended to specify that all states have this transition in common except for the states which are listed after the asterisk. Currently, most UML tools do not provide duplicate naming of states, so State Lists can only be rarely used. If isStateList is set to a true value, it implies that the name of the state represents multiple state names. Hidden from the user, state lists have to be decomposed into distinct states by the modeling tool as noted in Section 8.2.2.4.

7.3.3.2 DEFERRABLE TRIGGERS AND NOTATION

Events of signal reception which cannot trigger a transition are discarded. This is the default action. However, a state being in an active configuration can specify a set of deferred trigger. If a trigger is specified to be deferred by the active configuration state and no transition is enabled after the trigger is dispatched, the trigger is remains pending.

A trigger instance is pending as long it is deferred by the active configuration. This situation lasts until a configuration is reached where the trigger is not deferred anymore and is ready to be dispatched again or finally discarded. The dispatching mechanism is serializing the triggers to be dispatched in a sequence, because the step semantics assumes a single event dispatch. After executing the RTC step, the default dispatching mechanism issues the subsequent trigger from the input queue in a first-in-first-out (with priorities) manner. Hence, it is guaranteed that there is no conflict even if further triggers have been raised, because only one event will be dispatched.

So, all events that occur while the state machine remains in this state and are not able to trigger an enabled transition are not dropped, but saved until the next state has been reached. These are the *deferred triggers*. This is defined in the `deferrableTrigger` attribute. The notation for deferring triggers is the following:

```
<deferrableTriggers> ::= (<trigger> [ '\,' <trigger>]* |
                        '*' [ \'(<trigger> [ '\,' <trigger> ]* \)' )
                        \'/defer'
```

This leads to two complementary definitions of deferrable triggers: A positive list of all triggers that are to be deferred and a negative list of all triggers which are to be deferred with the exception of those that are specified in the set. The following expression provides an example:

```
trigger1, trigger2/defer
```

This means, the triggers *trigger1* and *trigger2* are deferred to the next state if they are caused by an event and cannot trigger an enabled transition in the active state.

```
*/defer
```

The asterisk defines that all triggers that cannot trigger an enabled transition are deferred to the next state.

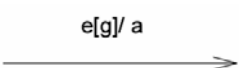
```
*(trigger1, trigger2)/defer
```

This negation expression defines that triggers which cannot trigger an enabled transition with the exception of the *trigger1* and *trigger2* are deferred.

When events are queued that cannot trigger a transition, only a silent discard or a deferring of this event is possible. An exception or a state machine termination does not occur which is a semantic variation point in the UML.

7.3.4 Transition

A transition is extended with the stereotype `<transition>` from the metaclass `TRANSITION` (FROM `BEHAVIORSTATEMACHINES`). A transition is a directed link between a source and a target vertex. A transition may be part of a *compound transition*. A compound transition represents the complete path of all executed transitions as the reaction of the state machine to the occurrence of an event of a particular type. It takes the state machine from one state configuration to another while crossing potential pseudo states in-between, such as decision or merge nodes.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Transition		15.3.14 Transition
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	<i>TAGGED VALUES</i>
«transition»	Transition (from BehaviorStateMachines)	priorized: Boolean = false
<i>CONSTRAINTS</i>		<i>COMMENTS</i>
context Transition inv: self.source->oclIsKindOf(State) implies (self.trigger->notEmpty() or self.guard->notEmpty()) inv: self.source->oclIsKindOf(Pseudostate) implies self.trigger->isEmpty() inv: self.source->oclIsTypeOf(merge) implies self.guard->isEmpty() inv: self.source->oclIsTypeOf(start) implies self.guard->isEmpty() inv: self.trigger.event->oclIsKindOf(SignalEvent) implies not self.trigger.event.signal-> oclIsKindOf(signalList) inv: self.redefinedTransition->isEmpty() inv: not self.source.outgoing->reject(self)-> collect(trigger)->collect(event)-> exist(self.trigger.event)		Both trigger and guard must not be empty at the same time except after the start state (initial pseudostate). A signalList shall not be specified in a SignalEvent. A transition cannot be redefined. There must not be a transition defined that has the same originating state and the same event trigger defined.

The UML Superstructure document [OMG06] introduces alternative notations for each of the attributes trigger, guard and effect of the Transition metaclass. Current UML tools only allow the specification of a behavior diagram which are state machine diagrams, activity diagrams and interaction overview diagrams. To the best of one's knowledge, the only UML 2 tool supporting a transition-centric view of a state machine is currently the UML tool Tau G2 [Tau].

The effect attribute of the transition is an association to an instance of the Behavior metaclass, see Figure 66 for the abstract syntax. This figure shows the metaclass associations of the Transition metaclass. The Statemachine and Activity metaclass are specialized from the Behavior metaclass that defines the effect of the Transition metaclass.

Since Statemachine and Activities are specialized metaclasses of Behavior, the effect of a Transition can be specified by a Statemachine or by an Activity. Note that an Interaction is a specialization of Behavior as well. However, this metaclass is not supported for behavior description by this profile. Therefore, there are various kinds of notations possible which are conformant to this UML CS profile. This is also caused by the scope of this profile. Most UML 2 tools available today do not support any transition notation other than textual expressions for the trigger, the guard or by reference to an activity or state machine or interaction.

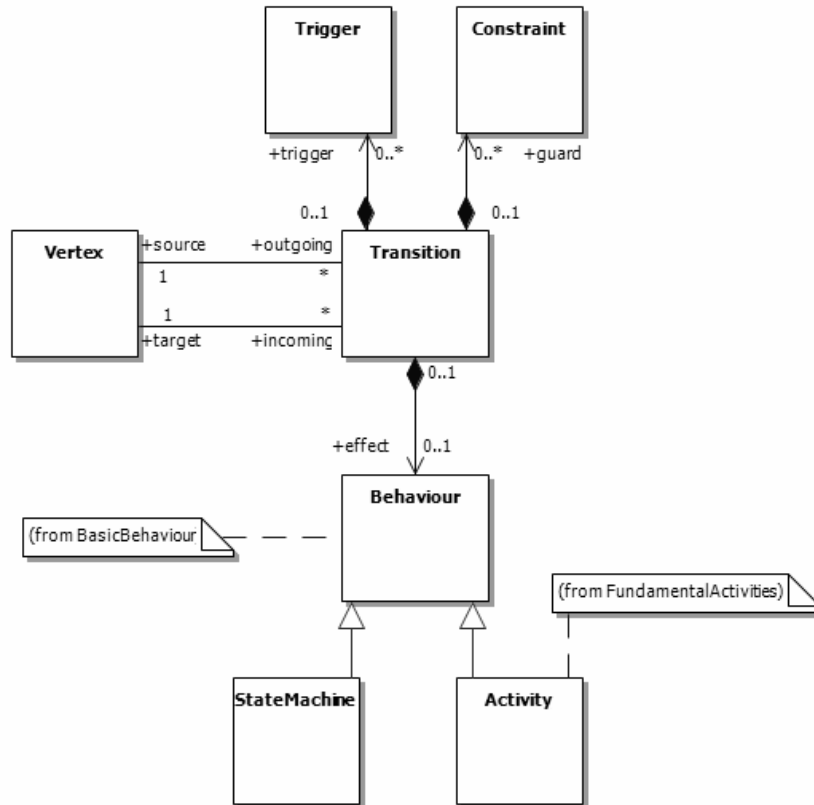


Figure 66: Extract of the Transition Metaclass with respect to Behavior

Alternative Notations

This profile supports four different notational styles which are compliant to the abstract syntax specified by the UML Superstructure. The first one is the *textual notation*. That is, the Transition attributes trigger, guard and effect are specified by an EBNF defined textual style:

```

<transition> ::= <trigger> [ '\,' <trigger> ] *
                [ '[' <guard expression> ']' ] '\/'
                <behavior expression>
  
```

The trigger is the trigger causing the execution of this transition if the guard expression is satisfied. After this, the behavior expression, defined as the effect, will be executed. The Behavior can be specified by Activities. The textual notation for activities is covered in detail in Section 7.4. A textual notation to define state machines is not supported. See the following Figure 67 for an example of a textual notation of a transition. Following the EBNF rule above for the concrete syntax, the transition is specified by the expression

```

sig1(i) [g>3] / i:=i+1; OUTPUT sig1(i); set(timer1,now+2);
  
```

This transition expression is composed by the trigger, guard and behavior-expression. Separating the composed expression to its components, the result is:

```

<trigger> ::= 'sig1(i)'  

<guard expression> ::= 'g>3'  

<behavior expression> ::= 'i:=i+1; OUTPUT sig1(i); set(timer1,now+2);'

```

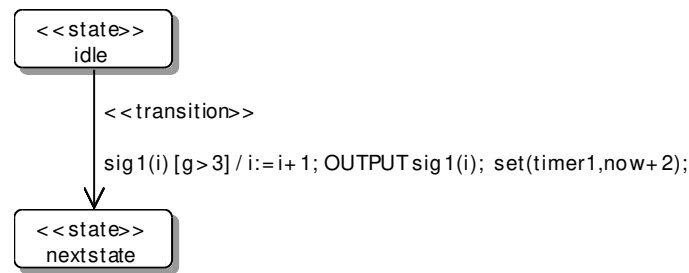


Figure 67: Textual Notation for Transition

The next alternative notation is the *mixed-textual* notation. This is described also in the UML Superstructure document as being an alternative notation for transitions. This notation introduces three different graphical elements for the expression of triggers combined with guards for sending signals and for specifying Activities in textual notation which is depicted in the following Figure 68.

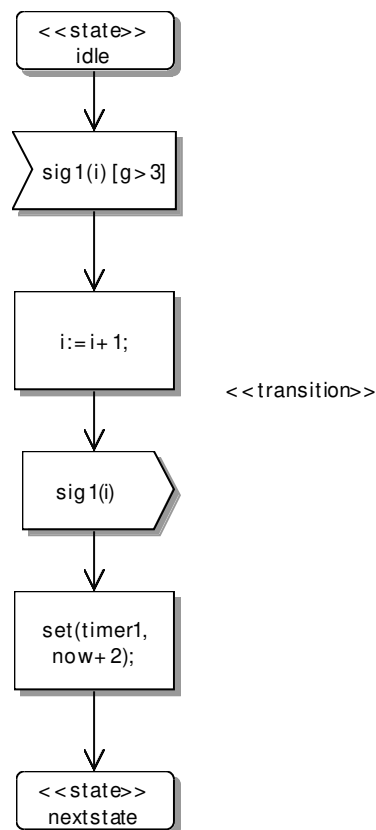


Figure 68: Mixed-textual Notation for Transition

The graphical elements used in this notation are shown for the trigger and guard part on the left in Figure 69. The textual notation for the mixed-textual notation is the same as for the pure textual notation with the exception that the behavior-expression is omitted. The behavior-expression is shown in the graphical element, an action box on the right.

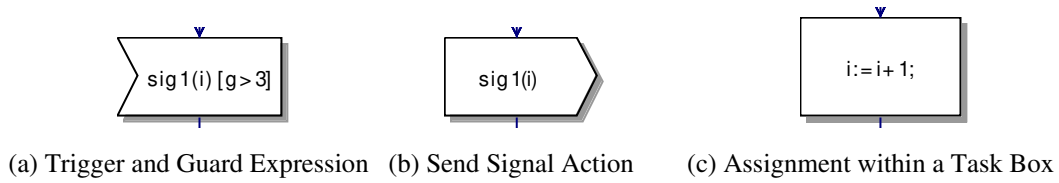


Figure 69: Graphical Elements for Mixed-textual Notation

An action box specifies the behavior in textual notation. An exception is applied to the sending of a signal. This action has a separate graphical notation despite the fact that it can also be expressed within the action box symbol. This is the proposed notation by the Superstructure document and is applied here.

Another way to express a Transition is the *referenced* notation. If such a notation is used, the behavior-expression in a Transition references an activity or a state machine which is executed when the transition fires. This notation is quite common in most UML 2 tools.

The final notation for a Transition is the proposed notation by this UML profile: the *transition-centric graphical* notation. If the UML tool that is used supports such a notation, it should be used. Using this notation, the behavior elements to be executed are directly constructed together with their invoking transition. Therefore, state machine elements and the activity elements are used in a single diagram. Note that this does not collide with the UML Superstructure document, as UML does not define the terminology *diagram* for itself. It is neither required to separate those kinds of elements nor it is forbidden to merge both notations into one single representation. The following Figure 70 shows an example that is equivalent to the other transition notations given in this section.

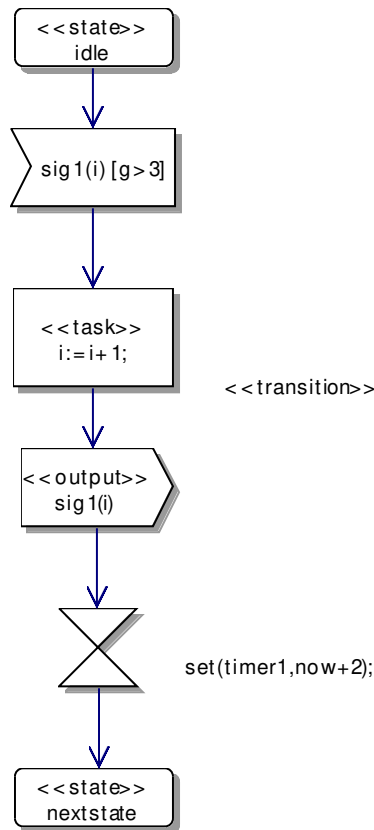


Figure 70: Transition-centric Notation of a Transition

Currently, it is not covered in the UML Superstructure document how the decomposition of a Transition into its alternative notation consisting of a receive signal event, a send signal and an action box is done. In particular, the relation of the send signal notation is unclear with respect to the Transition. The send signal element is being mapped to a SendSignalAction, but the abstract syntax is not noted.

However, in the transition-centric notation it is preferred to use of the receive signal element and graphical elements from the activities, covered in Section 7.4.

Semantics

A transition invokes a state change if its source state is active, one of its triggers is raised and the Boolean guard expression is satisfied. A trigger is invoked as the direct consequence of an event. Then the behavior specified by the transition's effect attribute is executed at an instant in a run-to-completion step. Informally, the semantics of a step involve the execution of a non-conflicting compound transition from an active, current state configuration.

Before a transition is triggered, at first, the guard expression is evaluated and then the trigger is consumed. Otherwise, the trigger is not consumed⁴ and remains pending.

The guard constraint specifies the condition that allows this transition to be selected. The trigger is associated with an event which defines the trigger to be raised to select this transition. The effect property defines the behavior that is executed when this specific transition has been fired.

7.3.4.1 TRANSITION SELECTION

The procedure of transition selection specifies the enabled transitions that will be executed. This procedure has to take into account two major considerations that affect the selection of a transition: conflicts and priorities.

In a given state, it is possible that more than one transition to be enabled within a state machine. The question then is which of these transition is to be executed. As an example, for two outbound transitions exiting a state s with one transition being labeled $t[g1]$ and the other $t[g2]$, with t being the trigger for both transitions, and both guards $[g1]$ and $[g2]$ are satisfied, only one transition can be executed. Two transitions conflict if they both exit the same state. However, the events that have invoked a trigger are collected in an input queue with FIFO semantics if the default scheduler is used for processes. The transition is selected whose trigger has been raised prior to all others. In addition, potential priorities of the triggers can also affect the transition selection. An internal transition within a state only conflicts with transitions that cause an exit from that state, but the internal transition concept is not supported. However, defining transitions being in conflict is supported by the UML CS profile using non-deterministic triggers or decisions.

⁴ Whether a trigger of a guarded transition whose condition evaluates to false is consumed or not, is not clearly described in the UML 2.1 Superstructure document [OMG06]. It is assumed that the described behavior here reflects the intended behavior.

7.3.4.2 PRIORITIES

A possible approach to resolve transition conflicts is to use priorities, but they cannot solve all conflicts. If a trigger has been raised by the occurrence of a signal event, the transition selection is as follows:

Only transitions can be selected which have their guards being satisfied. First, the transitions are considered which have the *priorized* attribute set to a true Boolean value. Then the transition is selected which has an event trigger pending by the reception of a signal with the highest priority value of the group. If there are multiple signals received which have the same priority value, the signal's trigger is used that has been received at first. If no transition with its *priorized* attribute set to true has been selected, the transitions with the *priorized* attribute set to false are considered. Analogously, a signal trigger is selected with the highest priority value or the first signal received if the priority values of the group are equal.

Furthermore, due to conflicts that may arise by state hierarchies it is necessary to derive priorities among conflicting transitions. Such a conflict is solved by the definition that a transition emanating from a sub-state has higher priority than a conflicting transition emanating from the containing states. To derive the priority, the source state of these transitions is decisive. Taking as an example that *t1* is a transition whose source state is *s1* and *t2* has source *s2*, then

- If *s1* is a sub-state of *s2*, then *t1* has higher priority than *t2*.
- If *s1* and *s2* do not comprise the other, there is no priority between *t1* and *t2*.

Using default scheduler, the transition that will fire is the first one received or with the highest specified priority that satisfies the following conditions:

- The transition is enabled.
- There are no pending conflicts with other transitions currently unresolved.

Intuitively, all the transitions with a lower priority will be disregarded.

7.3.5 Action Node (optional)

The Action Node contains textual instructions to be executed by the UML CS state machine. All actions and activities specified by UML CS are described in Section 7.4.

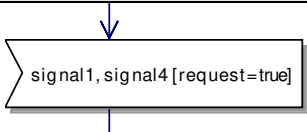
<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Action Node	<div style="border: 1px solid black; padding: 5px; display: inline-block;">MinorReq := Id;</div>	15.3.14 Transition
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
	Transition (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINS</i>

Semantics

The action node is an alternative graphical notation for the behavioral expression of a transition, as specified by the effect association. Therefore, the action node does not imply a different metaclass in the UML metamodel. This action node merges all actions which are specified for a behavior into a single model element in its textual representation. The rules for the textual notation are described in detail in Section 7.4.

7.3.6 Receive Signal (optional)

The Receive Signal node is a part of the alternative notation for a transition. The alternative notation for a transition separates the condition of a transition to fire from its triggered behavior. This Receive Signal node specifies the part triggers and guards. The Transition class has an association with the Trigger class. Note that in the UML 2.0, the Trigger metaclass defines a port attribute. That allowed to determine the port through which the specific event trigger has been invoked (corresponding to the *INPUT VIA* statement as noted in Section 5.2.2). With the UML 2.1, this port association has been removed thus disabling the way to retrieve such a port. To re-enable this feature, the SignalEvent metaclass has been extended with the missing tagged values, see Section 7.3.7.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Receive Signal Action		15.3.14 Transition
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
	Transition (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>

Semantics

If the event is a SignalEvent, this node is triggered by the reception of the specified input signal instance resulting in the *consumption* of the signal. When the input signal is consumed, it makes the information available conveyed by the signal. The variables associated with the received signal are assigned to the data items conveyed by the consumed signal. The data items are specified in the assignment specification of the SignalEvent and are successively assigned to the given variables from left to right. A variable can be left unspecified (optional) in the input node and the corresponding data item is simply discarded. If the signal does not define any data item with a type, the corresponding variable becomes *undefined*.

The *Process Identification* of the originating agent carried by the signal instance is assigned to the *sender* tag definition of the consuming agent. Signal instances coming from the environment to an agent instance within the system will always carry a *Process Identification* different from any agent in the system but always remains distinct. A Receive Signal Action may specify multiple signals to be received for consumption. If several inputs initiate the same transition, a list of signals which includes

timer signals may be used instead of specifying identical transitions for different inputs separately. Multiple signals as a signal list have to be separated by colons. Only a single signal specified in this signal list can be consumed if the transition is fired. The signal selected to be consumed is the one waiting in the process input queue as the next one that is ready for consumption.

The sender of a signal as well as the receiving port can be determined by the port attribute of the trigger. There are additional clauses introduced which respect these attributes. A trigger can optionally constrain the consumption of a signal to a distinct process identification from which this signal has been received. This can be done using the *FROM* clause. For example, a process with the Receive Signal Action textual notation

```
INPUT signal1(x) from self, signal2
```

would only consume the *signal1* with a variable *x* if it has been received from itself. Alternatively, the signal *signal2* received from any process would be consumed. Note that both signals cannot be consumed even if both signals have been received and are placed in the process signal input queue. Only the single signal is consumed which is to be selected as next for consumption. The sender's process identification can be retrieved from the SignalEvent Signal association which conveys the Pid within its *signal* tagged value.

A Receive Signal Action can optionally constrain the consumption of a signal to a distinct port using the *VIA* clause. For example, a process with the «input» instruction

```
INPUT signal1(x) via port1, signal2 from self, signal3
```

would only consume the *signal1* with a variable *x* if it has been received via a port labeled *port1*. Alternatively, the signal *signal2* would be consumed if it has been received from itself or the signal *signal3* would be consumed. Note that only one signal of these signals can be consumed if the transition is fired.

Alternative Notations

The Receive Signal Action symbol notation is currently not supported by UML 2 tools available at the time of writing (mid-2006) with the exception of Tau G2. Therefore, alternative textual notations are supported if such a tool is being used. The first method is to use a «transition» - a stereotype extending the state machine's transition class. It is described later in detail. The second method is by using an Action box if it is available in the UML tool. The textual notation complies with the UML notation for a transition which is available within a «transition»:

```
«transition» ::= <trigger> [ <guard> ] [ 'priority' ]
<trigger>      ::= <triggerclause> [ ',' <triggerclause> ]*
<triggerclause> ::= <trigger identifier> [ 'from' <Pid expression> |
                          'via' <port identifier> ]
<guard>       ::= '[' <Boolean expression> ']'
```

Within an «action» box, the following textual notation has to be used:

```
«action»      ::= 'INPUT' «transition»
```

7.3.7 SignalEvent

SignalEvent is extended with the stereotype «signalEvent» from the metaclass SIGNAL_EVENT (FROM COMMUNICATION). This stereotype extends the SignalEvent metaclass in a way that it allows the restriction of the port through which a signal has been received or the signal sender's process identification to a specific value.

UML NODE TYPE	UML NOTATION	REFERENCE
SignalEvent	- no specific notation -	13.3.25 SignalEvent
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«signalEvent»	SignalEvent (from Communications)	from: Pid [0..1] via: Port [0..1]
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context signalEvent inv: self.from->size()<1 and self.via->size()<1		Both <u>from</u> and <u>via</u> tagged values must not be set at the same time.

Semantics

The «signalEvent» allows constraining the reception of a SignalEvent in two different ways. First, the port can be specified by means of the tagged value via through which a specific signal has to be received to enable this signalEvent. If the signal has been received through another port than being specified, this signalEvent will not trigger a transition. Second, the sender of a signal can be constrained. If this is done by the from tagged value, this signalEvent will only trigger a transition if the sender's Pid and the specified from value match.

7.3.8 Composite State

A Composite State is extended with the stereotype «compositeState» from the metaclass STATE (FROM BEHAVIORSTATEMACHINES). A Composite State defines a set of states within a state. This allows decomposition of states into high detailed sub-states and activities.

UML NODE TYPE	UML NOTATION	REFERENCE
Composite State		15.3.11 State
UML CS STEREOTYPE	UML METACLASS	
«compositeState»	State (from BehaviorStateMachines)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS

<pre> context State inv: self.doActivity->isEmpty() inv: self.entryActivity->isEmpty() inv: self.exitActivity->isEmpty() inv: self.submachine->isEmpty() inv: self.regions->size()=1 inv: self.isComposite=true inv: self.isOrthogonal=false inv: self.isSimple=false inv: self.isSubmachineState=false inv: self.connection->isEmpty() inv: self.stateInvariant->isEmpty() inv: self.redefinedState->isEmpty() </pre>	<p>State Lists are not allowed for a composite state. Composite States cannot be redefined.</p>
--	---

Semantics

A «compositeState» contains a set of sub-states and exactly one region. Concurrency is not supported within a composite state. A region may have an initial node and a final node. Entry Point and exit points allow a connection between internal transition and external transition. The execution of a composite state starts when the containing state is entered. If the entry of the composite state is not done via an entry point, the behavior is invoked at the initial node. If the entry to the composite state is done via an entry point, the associated internal transition originating from the selected entry point is taken.

The connectionPoint defines the entry and exit points of this composite state. The deferrableTrigger defines the triggers to be deferred for the composite state including all sub-states. This applies analogously to the sub-states as all sub-states also have the same deferrableTrigger set definition.

7.3.9 Entry Point

The Entry Point pseudostate is extended with the stereotype «entryPoint» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES).


<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Entry Point	again ○	15.3.8 Pseudostate
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«entryPoint»	Pseudostate (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
<pre> context Pseudostate inv: self.kind=PseudostateKind::entryPoint inv: self.incoming->size()=0 inv: self.outgoing->size()=1 </pre>		

Semantics

An «entryPoint» pseudostate is an entry point of a composite state. It has a single transition to a vertex within the composite state. The name defines the name of this pseudostate.

7.3.10 Exit Point

The Exit Point pseudostate is extended with the stereotype «exitPoint» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES).


<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Exit Point		15.3.8 Pseudostate
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«exitPoint»	Pseudostate (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Pseudostate inv: self.kind=PseudostateKind::exitPoint		

Semantics

An «exitPoint» pseudostate is an exit point of a composite state. Entering an exit point of the composite state implies exiting of this composite state. It triggers the transition that has this exit point as source in the state machine containing the composite state of the «exitPoint». The name defines the name of this pseudostate.

7.3.11 Final State

The Final State is extended with the stereotype «finalState» from the metaclass FINALSTATE (FROM BEHAVIORSTATEMACHINES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Final State		15.3.2 FinalState
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«finalState»	FinalState (from BehaviorStateMachines)	
<i>CONSTRAINTS</i>		<i>COMMENTS</i>
context FinalState inv: (self.container->notEmpty() implies self.container.state->notEmpty()) or		The node must have least one <u>incoming</u> and no <u>outgoing</u> transitions. A finalState is only allowed within a composite state or within a

(self.region.statemachine.specification->oclIsKindOf(Operation))	state machine implementing an operation.
--	--

Semantics

The «finalState» can only be used within a composite state to end explicitly the composite state's behavior or an operation. The execution within the composite state or operation is then finalized and the execution is returned to the containing state or invoking process. The calling or owning process itself is not terminated.

7.3.12 Decision

The Choice pseudostate is extended with the stereotype «decision» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES). A «decision» node is used to split a single transition in multiple transitions. This allows a fine-grained control of the control flow of the state machine execution such that the decision, which path is taken, may be a result of previous actions that have been executed within the same run-to-completion step.

UML NODE TYPE	UML NOTATION	REFERENCE
Choice Pseudostate		15.3.8 Pseudostate
UML CS STEREOTYPE	UML METACLASS	
«decision»	Pseudostate (from BehaviorStateMachines)	
OCL CONSTRAINTS	INFORMAL CONSTRAINTS	
context Pseudostate inv: self.kind=PseudostateKind::choice inv: self.incoming->size()>=1 inv: self.outgoing->size()>=2 inv: self.outgoing-> forAll(t t.trigger->isEmpty() and t.guard->notEmpty()) inv: self.outgoing-> select(t t.guard.specification="else")-> size()<=1	During mapping from UML CS to SDL the choice conditions on the outgoing transition guards must be split into the common <i>question</i> and multiple different <i>answer</i> . There shall be only one <u>outgoing</u> transition with an “else” guard. The outgoing transition must not have an empty trigger and a guard.	

Semantics

When a «decision» node is reached, it results in the dynamic evaluation of the guards of the triggers of its outgoing transitions. This realizes a dynamic conditional branch. That is, the decision whose transition will be selected is chosen during the execution of the system. A static conditional branch (which is available by a junction element in UML – but not supported in this way in UML CS, see Section 7.3.18) specifies that the execution path is already determined prior to the firing of the first transition. This is contrary to a dynamic decision where the guards are only evaluated when the

transition to this decision pseudostate is being taken. This implies that the selection of a transition from this decision depends on the activities which have been executed up to this point.

If more than one of the guards evaluates to true, a transition is selected non-deterministically. If none of the guards evaluates to true, the model is considered ill-formed. A pre-defined *else* guard is available yielding a true value if all other transition guards yield false.

A non-deterministic decision can be implemented if all guards on the outgoing transition have a true value. A variable shall not be contained in the guard's expression.

7.3.13 Process Start

The Initial pseudostate is extended with the stereotype «start» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES). The «start» state represents the first state for the beginning of a process or composite state execution. Within a state machine (as multiple regions are not supported) and within a distinct composite state there can be at most one initial vertex.


UML NODE TYPE	UML NOTATION	REFERENCE
Initial Pseudostate	●	15.3.8 Pseudostate
UML CS STEREOTYPE	UML METACLASS	
«start»	Pseudostate (from BehaviorStateMachines)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context Pseudostate inv: self.kind=PseudostateKind::initial inv: self.incoming->size()=0 inv: self.outgoing->size()=1 inv: self.outgoing->forAll(t t.trigger->isEmpty() and t.guard->isEmpty())		The start pseudostate <u>kind</u> shall be <i>initial</i> . The node shall have no <u>incoming</u> and exactly one <u>outgoing</u> transition. There shall be exactly one single «start» node within a state machine. The transition connected to this start node shall have an empty trigger and an empty guard.

Semantics

The outgoing transition from the «start» node may have a behavior like all transitions. However, the outgoing transition shall not specify any trigger or guard. Therefore, the next transition has to be taken right after the instantiation of the containing agent.

7.3.14 Process Stop

The Terminate pseudostate is extended with the stereotype «stop» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES). When a «stop» is entered, it implies that the execution of this state machine is terminated by means of its context object.


<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Terminate Node		15.3.8 Pseudostate
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«stop»	Pseudostate (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Pseudostate inv: self.kind=PseudostateKind::terminate inv: self.incoming->size()>=1 inv: self.outgoing->size()=0		The stop pseudostate <u>kind</u> must be <i>Terminate</i> . The node must have one or more <u>incoming</u> and no <u>outgoing</u> transition.

Semantics

When a «stop» is entered, it implies that the execution process of this state machine has ended and by means of its context object is terminated. The state machine does not perform any actions from now on. When reaching this node, it neither exits any states nor performs any exit actions other than those associated with the transition leading to «stop». The associated process is terminated as well. A re-start is not possible. Signals being received after the termination are still being queued, but not consumed. As there is no response to a sent signal, the sender of the signal to this process is not notified of the reception of its signal by a terminated process.

7.3.15 History

The ShallowHistory pseudostate is extended with the stereotype «history» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Pseudostate		15.3.8 Pseudostate
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«history»	Pseudostate (from BehaviorStateMachines)	
<i>CONSTRAINTS</i>		<i>COMMENTS</i>
context Pseudostate inv: self.kind=PseudostateKind::shallowHistory inv: self.incoming->size()>=1 inv: self.outgoing->size()=0		The history pseudostate <u>kind</u> must be <i>ShallowHistory</i> . The node must have least one <u>incoming</u> and no <u>outgoing</u> transitions.

Alternative notation

An alternative notation is the SDL like notation for a «history» pseudostate. For this, if a name of a state is equal to '-', which is the <minus> sign, it is mapped to a «history» shown in the following Figure 71:



Figure 71: Alternative Notation for History State

Semantics

A «history» represents a shallow history node: it is a replacement for the most recent active state of a state machine or sub-state of its containing state (but not the sub-states of that sub-state). A composite state can have at most one «history» node. This node is equivalent to a transition that is connected to the most recent active sub-state of a state. At most one transition may originate from the history node to the default history state which is taken in the case where the composite state has not been active before.

7.3.16 Method Start

The Method Start is extended with the stereotype «methodStart» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES).


UML NODE TYPE	UML NOTATION	REFERENCE
Initial Pseudostate	●	15.3.8 Pseudostate
UML CS STEREOTYPE	UML METACLASS	
«methodStart»	Pseudostate (from BehaviorStateMachines)	
CONSTRAINTS		COMMENTS
context Pseudostate inv: self.kind=PseudostateKind::initial inv: self.incoming->size()==0 inv: self.outgoing->size()==1 inv: self.container->size()==1 implies self.container.region.subvertex-> exist(v isStereotyped(v,start))		The methodStart pseudostate <u>kind</u> must be <i>Initial</i> . The node must have no <u>incoming</u> and exactly one <u>outgoing</u> transition. There shall be no start node defined in the same region.

Semantics

The «methodStart» is the initial pseudostate for a method to start. There shall be exactly one initial state defined within the same region. The invocation of a method can be done by a received CallEvent or CallBehaviorAction. The BehavioralClassifier is defined by its associated Operation.

7.3.17 Method Return

The Method Return is extended with the stereotype «methodReturn» from the metaclass FINALSTATE (FROM BEHAVIORSTATEMACHINES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Final State		15.3.2 FinalState
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«methodReturn»	FinalState (from BehaviorStateMachines)	
<i>CONSTRAINTS</i>		<i>COMMENTS</i>
context FinalState inv: self.incoming->size()>=1 inv: self.outgoing->size()=0		The node must have at least one incoming and no outgoing transitions.


Semantics

The «methodReturn» is the final state of a method. When this node is reached by a transition, all variables with out- or return-direction are assigned to their final value. The method is then terminated and the execution is returned to the calling process.

7.3.18 Merge

The Junction pseudostate is extended with the stereotype «merge» from the metaclass PSEUDOSTATE (FROM BEHAVIORSTATEMACHINES).

In the UML, a Junction is a pseudostate that is used to split an incoming transition into multiple outgoing transition segments with different guard conditions. A Junction pseudostate is an element that represents a *static conditional branch* while the Choice pseudostate represents a *dynamic conditional branch*, see Section 7.3.12 for a detailed explanation. Static conditional branches are currently not supported, so this node is a notational replacement to tie multiple transitions together. Therefore, a guard constraint is not allowed on the outgoing transition.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Junction Pseudostate		15.3.8 Pseudostate
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«merge»	Pseudostate (from BehaviorStateMachines)	
<i>OCL CONSTRAINTS</i>	<i>INFORMAL CONSTRAINTS</i>	
context Pseudostate inv: self.kind=PseudostateKind::junction	The merge pseudostate <u>kind</u> must be <i>junction</i> . It is only allowed to merge	

inv: self.incoming->size()>1 inv: self.outgoing->size()=1 inv: self.outgoing->select(t t.guard->isEmpty()) ->isEmpty()	multiple <u>incoming</u> transitions into a single <u>outgoing</u> transition. The outgoing transitions shall not have guards.
--	--

Semantics

A «merge» is used to tie multiple incoming transitions into a single outgoing transition representing a subsequent shared transition path. A guard is not allowed on the outgoing transition. The node itself has no semantics.

7.4 Activities

Besides of state machines, a behavior in UML can be specified by means of activities. An activity is a collection of actions, possibly specified within an activity diagram. The control flow of the actions is transitive. That is, each action has its successors. A very good overview on the UML 2 action and activity models can be found in the series [Boc03a, Boc03b, Boc04a, Boc04b, Boc05].

The UML 2 does not provide a specific syntax for actions and activities. Therefore, this profile introduces alternative concrete notations in a graphical or textual representation. The recommended representation is the graphical notation. However, several actions are only available in a textual representation and shall be specified within a task or action box or an `OpaqueAction`, for instance.

The following metaclasses of the UML are extended in this section: `Activity`, `SequenceNode`, `ControlFlow`, `CallOperationAction`, `CreateObjectAction`, `SendSignalAction`, `CreateObjectAction`, `WriteStructuralFeatureAction`, `WriteVariableAction`, `DecisionNode`, `MergeNode`, `ConditionalNode`, `LoopNode`, `OpaqueAction`, `InitialNode` and `ActivityFinalNode`.


7.4.1 Activity

The Activity is extended with the stereotype «activity» from the metaclass `ACTIVITY` (FROM `BASICACTIVITIES`, `COMPLETEACTIVITIES`, `FUNDAMENTALACTIVITIES`, `STRUCTURESACTIVITIES`). An «activity» is the *effect*-activity of a transition (see Section 7.3.4) or the method of an operation. An activity describes a part of a behavior by means of a control and data flow model. An activity is modeled as activity nodes that are connected by activity edges. These edges represent the possible flow of execution. An activity node can also be executed as a subordinate behavior from other activities. For example, it might be invoked by an arithmetic computation, a call to an operation or a manipulation of objects. An activity can also fork in multiple control flows according to UML standard, but this feature is not supported in UML CS. Hence, features like synchronization and concurrency control are not possible within an activity. However, other flow-of-control constructs like decisions are available.

Activities can invoke further sub-activities which ultimately resolve into individual actions. In most cases, activities are invoked indirectly, e.g. as methods which are bound to operations that are directly invoked. Activities may describe procedural computation. Activities can describe the methods which define the behavior of operations of classes. An activity may contain the following kinds of actions:

- Invocations and occurrence of a primitive functions and behavior such as activities or arithmetic functions,
- Communication actions by sending signals or invoking remote operations,
- Modification of objects and their properties by reading or writing attributes.

In an activity, the contained actions have no further decomposition. However, the execution of a single action element can result in the execution of several further actions. For example, this is the case for a method call action. It invokes an operation implemented by an activity comprised of actions that execute before the call action can complete.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Activity		12.3.4 Activity
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«activity»	Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuresActivities)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Activity inv: self.isReadOnly=false inv: self.isSingleExection=true inv: self.redefinedBehavior->isEmpty() inv: self.node->size()<2 inv: self.node->size()=1 implies self.node[0]->oclIsKindOf(SequenceNode) inv: self.handler->isEmpty()		One SequenceNode shall exist. This prohibits token parallelism from beginning. An exception handler shall be empty, as exceptions are not supported currently.

Textual notation

```
«activity» ::= 'method' <activity name>
            <procedure formal parameters> «begin»
<activity name> ::= <procedure name>
```

Semantics

Activities provide a means to model a behavior in a visual manner. While state machines only provide the state-based viewpoint, the viewpoint itself is focused on the activities. Activities only model the behavior that is associated to a behavior effect on state machine transitions or to an operation of an agent or the operation of a data type definition.

In the UML, activity models have a control and data flow approach by initiating further behaviors, because in the moment others finish and when inputs are available they are started. In this profile, exactly one single control flow has to exist. This implies that multiple control flows cannot exist in parallel during run-time. Multiple initial nodes must not exist; control flow forks are not supported. The execution of an activity begins at the initial node and terminates at the activity final node. The node defines the contained actions. However, the only node allowed is a SequenceNode that enforces sequential execution. ownedParameter defines the parameters passed to the activity. variable defines all local variables.

7.4.2 Compound Statements

The SequenceNode is extended with the stereotype «sequenceNode» from the metaclass SEQUENCENODE (FROM STRUCTURED ACTIVITIES). A «sequenceNode» is a structured activity node that executes its actions in an ordered manner.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
SequenceNode	- No graphical notation -	12.3.47 SequenceNode
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«sequenceNode»	SequenceNode (from StructuredActivities)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context SequenceNode inv: self.executableNode->select(n n-> oclIsKindOf(initialNode))->size()<=1 inv: self.executableNode->select(n n-> oclIsKindOf(ForkNode))->isEmpty() inv: self.executableNode->select(n n-> oclIsKindOf(AcceptEventAction)) ->isEmpty()		There must be a maximum of one initial node present (this is where the activities begins executing). A ForkNode enables concurrency of control flows and therefore, is not supported. Waiting for events triggers is only allowed within state machine and is not allowed in activities.

Textual notation

```
«sequenceNode» ::= [ <comment body> ] <left curly bracket>
<statement list> <right curly bracket>
```

Semantics

A «sequenceNode» is a structured activity node that executes its contained actions in order. A sequence node has its own visibility scope. That means that local variables cannot be accessed from the outside of this sequence node. The executableNode defines the ordered collection of actions.

7.4.3 ControlFlow

The ControlFlow is extended with the stereotype «controlFlow» from the metaclass CONTROLFLOW (FROM BASICACTIVITIES). A «controlFlow» is a control flow transition that specifies the exact order of actions to be executed.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
ControlFlow	↓	12.3.19 ControlFlow
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	
«controlFlow»	ControlFlow (from BasicActivities)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context ControlFlow		A controlFlow may only have a guard

inv: self.activity->notEmpty() inv: self.source->oclIsKindOf(decisionNode) = self.guard->notEmpty()	specified if its <u>source</u> is a decisionNode. Otherwise, there shall be no guard defined.
---	--

Semantics

A «controlFlow» is an edge that starts the action that is connected at its end if the action at its source has finished. In contrast to transitions of a state machine, a controlFlow cannot specify any trigger.

7.4.4 Creating Objects

The CreateObjectAction is extended with the stereotype «createObject» from the metaclass CREATEOBJECTACTION (FROM INTERMEDIATEACTIONS). «createObject» is an action that creates an object from a given classifier and returns the object as the actions' result.

UML NODE TYPE	UML NOTATION	REFERENCE
CreateObjectAction	- No graphical notation -	11.3.16. CreateObjectAction
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«createObject»	CreateObjectAction (from IntermediateActions)	argument: InputPin[0..*]
OCL CONSTRAINTS	INFORMAL CONSTRAINTS	
context CreateObjectAction inv: isStereotypedBy(self.classifier, class) inv: isStereotypedBy(self.classifier, agent) inv: self.result->isEmpty() or self.result->oclIsKindOf(Pid)	The creation attempt shall only be applied on an agent or class type. The <u>result</u> attribute can be left un-assigned. In this case, only the internal variable offspring is set to the Pid of the new instance created. If a result is specified, the Pid of the new instance is also written to this pin.	

Textual Notation


```
«createObject» ::= [<identifier> `:='] `new' <agent identifier>
                `(' [<actual parameters>] `)' <semicolon>
```

Semantics

The new object is created and the classifier of the object is set to the given agent or classifier (class). For agents, the internal variable *offspring* is set to the agent's process id (Pid) value. The new object is returned as the result value of the action. Besides of the object creation, there is no other effect. The «createObject» action can be used for an active process as well as for a passive class. If an agent attempts to create more object instances than specified by the maximum number of instances in the agent definition, no instance is created. The offspring value of the creating agent is set Null and the execution continues. Extending the UML metaclass, this stereotype allows specification of actual parameters to the new object by means of the argument tag. The result is the new *Pid* of the created agent instance. The classifier specifies the instance type to be created.

7.4.5 Signal Output

The `SendSignalAction` is extended with the stereotype `«output»` from the metaclass `SENDSIGNALACTION` (FROM `BASICACTIONS`). An `«output»` is an action that creates a signal instance and transmits it to the target object.

UML NODE TYPE	UML NOTATION	REFERENCE
Send Signal Action		11.3.4 <code>SendSignalAction</code>
UML CS STEREOTYPE	UML CS METACLASS	TAGGED VALUES
<code>«output»</code>	<code>SendSignalAction</code> (from <code>BasicActions</code>)	<code>via: Port [0..1]</code> <code>dynamicPort: Integer</code>
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
<p>context <code>SendSignalAction</code></p> <p>inv: <code>self.target.InputPin ->isOclTypeOf(ValuePin)</code></p> <p>inv: <code>self.context=port.featuringClassifier</code></p> <p>inv: <code>not self.signal->oclIsKindOf(timer)</code></p> <p>inv: <code>not self.signal->oclIsKindOf(signalList)</code></p> <p>inv: <code>self.onPort->notEmpty()</code> implies <code>self.extension_output.via = self.onPort</code></p> <p>context <code>output</code></p> <p>inv: <code>self.via->notEmpty()</code> implies <code>self.via.requiredInterface.ownedReception->exists(r r.signal=self.signal)</code></p> <p>inv: <code>self.via->notEmpty()</code> implies <code>self.base_SendSignalAction.target->isEmpty()</code></p> <p>inv: <code>self.dynamicPort>=0</code></p> <p>inv: <code>self.port->size()=1</code> and not <code>self.port.isDynamic</code> implies <code>self.dynamicPort=0</code></p> <p>inv: <code>self.port->size()=1</code> and <code>self.port.isDynamic</code> implies <code>self.dynamicPort<self.port.instances</code></p>		<p>The attribute <code>via</code> constrains the channels where the signal specified in <code>self.effect.signal</code> can be sent. Only <code>«channels»</code> that connect to the specified port can be selected for sending. The type of the <code>via</code> attributes shall be of stereotype <code>«port»</code>. The type of the target input pin shall be a <code>ValuePin</code>. The port addressed in the <code>port</code> property shall have the same classifier context in which the invoking action is defined. If a <code>via</code> port is specified, it has precedence over the <code>target</code> <code>inputPin</code> attribute. If a <code>via</code> is specified, the corresponding port must specify the signal in its <code>requiredInterface.ownedReception</code> property. <code>via</code> is a derived value from the <code>onPort</code> property.</p> <p>A timer or <code>signalList</code> as an extension of <code>Signal</code> cannot be sent.</p> <p>The <code>dynamicPort</code> tag definition specifies the dynamic port number of the dynamic port set. <code>dynamicPort</code> shall be zero if the target port is no dynamic port.</p>

Textual Notation

```
«output» ::= 'OUTPUT' <signal list> [' ,' <signal list>]*
<signal list> ::= <signal identifier> [<actual parameters>]
                ('to' <Pid expression> | 'via' <port identifier>
                ['(' <dynamicport identifier> ')'] <semicolon>
```

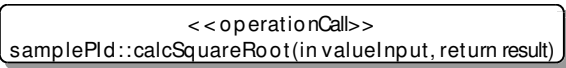

Semantics

An «output» is an action that creates a signal instance and transmits it to the target object that may be defined by the target attribute. The signal instance may cause the firing of a state machine transition and the execution of an activity. The argument attribute values which are associated to the signal are available to the execution of associated behaviors. The signal attribute defines the signal to be instantiated and sent. The sending process immediately continues execution. There is no reply message. An already created signal instance cannot be sent with this action.

The optional via tag definition specifies the port the signal has to be sent. The route of the signal from the originating process to the distinct target process must be unambiguously specified by the interfaces of the ports and channels' signal lists. If not, a target is arbitrarily chosen. The dynamicPort tag defines the index of the dynamic port set if the via specifies a dynamic port.

7.4.6 Call of Operations

The `OperationCall` is extended with the stereotype «operationCall» from the metaclass `CALLOPERATIONACTION` (FROM `BASICACTIONS`). The «operationCall» calls a visible method on the target process determined by the process identification or channel route.

UML NODE TYPE	UML NOTATION	REFERENCE
CallOperationAction		11.3.10 CallOperationAction
UML CS STEREO TYPE	UML CS METACLASS	TAGGED VALUES
«operationCall»	CallOperationAction (from BasicActions)	arguments: Property [0..*] via: port[0..1] dynamicPort: Integer isRemote: Boolean
OCL CONSTRAINTS	INFORMAL CONSTRAINTS	
<p>context CallOperationAction inv: self.operation.method-> oclIsKindOf(StateMachine) or oclIsKindOf(Activity) inv: self.onPort->notEmpty() implies self.extension_operationCall.via=self.onPort inv: self.extension_operationCall.isRemote = isRemote(self, self.operation)</p> <p>context operationCall inv: self.dynamicPort>=0 inv: self.port->size()=1 and not self.port.isDynamic implies self.dynamicPort=0 inv: self.port->size()=1 and self.port.isDynamic implies self.dynamicPort<self.port.instances</p>	<p>A behavior of an operation shall only be specified by means of a state machine or an activity.</p> <p>The invocation of a behavior can be constrained through a specific port by means of the <u>via</u> tag definition. <i>via</i> is derived from the <u>onPort</u> property.</p> <p><u>isRemote</u> indicates whether this operation call is local or remote (RPC) call.</p> <p>The <u>dynamicPort</u> tag definition specifies the dynamic port number of the dynamic port set. <u>dynamicPort</u> shall be zero if the target port is no dynamic port.</p>	

Textual Notation

```

«operationCall» ::= <procedure identifier> '(' <actual parameters> ')'
                  ('to' <Pid expression> | 'via' <port identifier>
                  ['(' <dynamicport identifier> ')']) <semicolon>

```

Semantics

«operationCall» is an action that transmits an operation call request to the target agent, where it may cause the invocation of associated behavior. The target attribute defines the target agent. All argument values of the action are passed as parameters to the invoked behavior. The operation call cannot call an operation that is out of its scope. This is evaluated based on the visibility modifiers of the target operation. Visibility rules are verified by static semantic checks and must not be violated during system execution.

This type of action is executed synchronously. That is, the execution of the call operation action is suspended until the execution of the invoked behavior completes and a reply signal is returned to the caller. Any values returned as part of the reply signal are available on the result output pins of the call operation action. When the reply signal is received, execution of the call operation action is complete.

An operation call can also be directed through a distinct port. This can be specified using the via attribute. According to the interface definitions on the agents, the operation call is conveyed through the possible channel to a target. The response of the operation call is also conveyed through the same path in the opposite direction.

If via or target is not empty, the modeling tool has to verify if the visibility of the target method does not violate the visibility rules defined in Section 7.2.5. If it complies, the modeling tools shall evaluate the isRemote property (checking whether this call is an remote procedure call). If true the modeling tool shall implicitly generate implicit (hidden to the user) channels, variables and signals according to the mapping rules provided in Section 10.5 in Z.100 [ITU02a].

7.4.7 Property Assignments

The WriteStructuralFeatureAction is extended with the stereotype «writeStructuralFeatureAction» from the metaclass WRITESTRUCTURALFEATUREACTION (FROM STRUCTURED ACTIONS).

UML NODE TYPE	UML NOTATION	REFERENCE
WriteStructuralFeatureAction	- No graphical notation -	11.3.53 WriteStructuralFeatureAction
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«writeStructuralFeatureAction»	WriteStructuralFeatureAction (from StructuredActions)	via: Port[0..1] dynamicPort: Integer isRemote: Boolean
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context WriteStructuralFeatureAction inv: self.value->oclIsKindOf(ValuePin) inv: self.extension_writeStructuralFeatureAction.		<u>isRemote</u> indicates whether the structural feature is accessed on a remote agent. The <u>dynamicPort</u> tag definition specifies

<pre>isRemote = isRemote(self, self.structuralFeature) context writeStructuralFeatureAction inv: self.dynamicPort>=0 inv: self.port->size()==1 and not self.port.isDynamic implies self.dynamicPort=0 inv: self.port->size()==1 and self.port.isDynamic implies self.dynamicPort<self.port.instances</pre>	<p>the dynamic port number of the dynamic port set. <u>dynamicPort</u> shall be zero if the target port is no dynamic port.</p>
---	---

Textual Notation

```
«writeVariableAction» ::= <assignment>
<assignment> ::= <variable> '=' <expression> ( 'to' <pid expression> |
    'via' <port identifier> [ '(' <dynamicport identifier>
    ')' ] <semicolon>
```

Semantics

A «writeStructuralFeatureAction» is used to modify attribute properties of an agent. It is an assignment action to a value of an agent variable. value attribute defines the value, the structuralFeature defines the property to be written. via specifies the port through which a remote variable on the target agent may be accessed.

First, the modeling tool has to verify if the visibility of the target structuralFeature does not violate the visibility rules defined in Section 7.7.3. If it complies the modeling tools shall evaluate the isRemote property (checking whether this call is a remote procedure call). If true the modeling tool shall implicitly generate implicit (hidden to the user) channels, variables and signals according to the mapping rules provided in Section 10.6 in Z.100 [ITU02a].

7.4.8 Local Assignment

The WriteVariableAction is extended with the stereotype «writeVariableAction» from the metaclass WRITEVARIABLEACTION (FROM STRUCTURED ACTIONS). A «writeVariableAction» is used to assign values to a variable.

UML NODE TYPE	UML NOTATION	REFERENCE
WriteVariableAction	- No graphical notation -	11.3.54 WriteVariableAction
UML CS STEREOTYPE	UML METACLASS	
«writeVariableAction»	WriteVariableAction (from StructuredActions)	
OCL CONSTRAINTS	INFORMAL CONSTRAINTS	
context WriteVariableAction inv: self.value->oclIsKindof(ValuePin)	The variable of the assignment shall be local.	

inv: not isRemote(self, self.structuralFeature)	
---	--

Textual notation

«writeVariableAction» ::= <assignment>
--

Semantics

A «writeVariableAction» is used to assign values to a local variable. The value attribute defines the value; the variable defines the variable to be written.

7.4.9 Task

The OpaqueAction is extended with the stereotype «task» from the metaclass OPAQUEACTION (FROM BASICACTIONS). The «task» is introduced for textual statements for the specific target language.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
OpaqueAction	- No graphical notation -	11.3.26 OpaqueAction
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«task»	OpaqueAction (from BasicActions)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context OpaqueAction inv: self.language->forall(l l="SDL") inv: self.language->size() = self.body->size()		

Textual Notation

«task» ::= <statements>

Semantics

This statements entered in a «task» are mapped to the target language without any conversion or interpretation. If the target language is SDL, the statements shall be of SDL-2000 syntax and semantics.

7.4.10 If

The ConditionalNode is extended with the stereotype «if» from the metaclass CONDITIONALNODE (FROM COMPLETESTRUCTUREDACTIVITIES, STRUCTUREDACTIVITIES).

UML NODE TYPE	UML NOTATION	REFERENCE
ConditionalNode	- No graphical notation -	12.3.18 ConditionalNode
UML CS STEREOTYPE	UML METACLASS	
«if»	ConditionalNode (from CompleteStructuredActivites, StucturedActivities)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context ConditionalNode inv: self.isAssured=true inv: self.isDeterminate=true inv: self.clause->size()=1 or self.clause->size()=2 inv: self.clause->forAll(c c.test->size()=1) inv: self.clause->forAll(c c.body->size()=1 and isStereotyped(c.body,sequenceNode)) inv: self.clause->exist(c c.test->isOclKindOf(Boolean)) inv: self.clause->size()=2 implies self.clause->exist(c c.test = “Always results a true value”)		The modeler has to assert that exactly one test will succeed in all «if» clauses. An if shall only specify one or two <u>clauses</u> . The informal definition of the <u>test</u> part reflects the else-part definition in Superstructure document.

Textual notation

```

«if» ::= 'if' '(' <Boolean expression> ')' <consequence statement>
        [ 'else' <alternative statement> ] <semicolon>
<consequence statement> ::= <statement>
<alternative statement> ::= <statement>

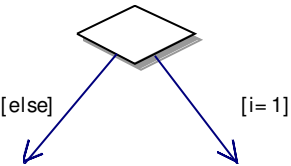
```

Semantics

«if» realizes a conditional branch within the specification of activities. The «if» stereotype implies from all clauses specified, exactly one clause will evaluate to true and its body will be executed. The evaluation of clauses and the execution of the body of a clause are deterministic.

7.4.11 Decision

The DecisionNode is extended with the stereotype «decisionNode» from the metaclass DECISIONNODE (FROM INTERMEDIATEACTIVITIES).

UML NODE TYPE	UML NOTATION	REFERENCE
DecisionNode		12.3.22 DecisionNode
UML CS STEREOTYPE	UML METACLASS	
«decisionNode»	DecisionNode (from IntermediateActivities)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
<pre> context DecisionNode inv: self.decisionInput->isEmpty() inv: self.incoming->forall(c c.oclIsKindOf(ControlFlow)) inv: self.outgoing->forall(c c.oclIsKindOf(ControlFlow)) inv: self.outgoing-> select(c c.guard.specification="else") ->size()<=1 </pre>		There shall be no behavioral input to the <u>guard</u> specification on the edges <u>outgoing</u> from this node.

Textual notation

```

«decisionNode» ::=  'decision' '(' <question> ')' [ <comment body> ]
<left curly bracket> <decision statement body> <right curly bracket>
<decision statement body> ::=
    <algorithm answer part>+ [ <algorithm else part> ]
<algorithm answer part> ::=
    '(' <answer> ')' <colon> <statement> <semicolon>
<algorithm else part> ::=
    'else' <colon> <alternative statement> <semicolon>
<alternative statement> ::= <statement>

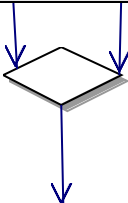
```

Semantics

A «decisionNode» realizes a conditional branch within the specification of activities. This is a switch-statement. The «decisionNode» stereotype implies that the clauses either have a static or a dynamic expression part (the question part) and a static expression part (the answer part). Both parts are compared for equality. All dynamic expression parts of the clauses must match whereas the answer parts may differ with the exception of the pre-defined else guard. If there is an *else* guard specified, there shall be only one being connected to the decisionNode.

7.4.12 Merging Controlflows

The MergeNode is extended with the stereotype «mergeNode» from the metaclass MERGENODE (FROM INTERMEDIATEACTIVITIES). A mergeNode merges several control flow edges to a single one.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
MergeNode		12.3.36 MergeNode
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«mergeNode»	MergeNode (from IntermediateActivities)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context MergeNode inv: self.activity->notEmpty() inv: self.outgoing->size()=1 inv: self.incoming->size()>0		Only one <u>outgoing</u> control flow must exist. This is required as UML also supports shared merge and decision symbols – this is not allowed here.

Semantics

«mergeNode» merges several control flow edges together into a single one. No synchronization is performed as there can be only be one single control flow be active within an activity. This node is used to merge control flows that, for instance, have been explicitly forked by a decisionNode or similar nodes.

7.4.13 For

The LoopNode is extended with the stereotype «for» from the metaclass LOOPNODE (FROM COMPLETESTRUCTUREDACTIVITIES, STRUCTUREDACTIVITIES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
LoopNode	- No graphical notation -	12.3.35 LoopNode
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	<i>TAGGED VALUES</i>
«for»	LoopNode (from CompleteStructuredActivites, StructuredActivities)	forAction: SequenceNode[0..1]
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context LoopNode inv: self.isTestedFirst=true inv: self.setupPart->size()<=1 inv: self.setupPart->size()=1 implies isStereotyped(self.setupPart, sequenceNode) inv: self.test->size()<=1		The <u>forAction</u> tag definition defines the activity to be executed each time when the <u>bodyPart</u> 's execution is completed.

inv: self.test->size()=1 implies isStereotyped(self.test, sequenceNode) inv: self.bodyPart->size()=1 inv: isStereotyped(self.bodyPart, sequenceNode) inv: self.result->isEmpty() inv: self.loopVariableInput->isEmpty()	
--	--

Textual notation

```

«for» ::= <for header> '{' <for body> '}' <semicolon>
<for header> ::= 'for' '(' [<for initialization>] ';'
                [<for condition>] <semicolon> [<for action>] ')'
<for initialization> ::= <statement>
<for condition> ::= <Boolean expression>
<for action> ::= <statement>
<for body> ::= <statements>

```

Semantics

«for» is a construct to provide iterations of activities by means of a running variable. Before the body is executed, the exit condition is tested first. The setupPart defines the initialization part which is executed before any run of the inner loop. It is only done once. The bodyPart defines the inner loop behavior. The test and decider attributes define the exit condition and decide on the abortion of the loop.

7.4.14 While

The LoopNode is extended with the stereotype «while» from the metaclass LOOPNODE (FROM COMPLETESTRUCTUREDACTIVITIES, STRUCTUREDACTIVITIES).

UML NODE TYPE	UML NOTATION	REFERENCE
LoopNode	- No graphical notation -	12.3.35 LoopNode
UML CS STEREO TYPE	UML META CLASS	
«while»	LoopNode (from CompleteStructuredActivites, StructuredActivities)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context LoopNode inv: self.isTestedFirst=true inv: self.setupPart->size()<=1 inv: self.setupPart->size()=1 implies isStereotyped(self.setupPart, sequenceNode) inv: self.test->size()<=1 inv: self.test->size()=1 implies isStereotyped(self.test,sequenceNode)		

inv: self.bodyPart->size()=1 inv: isStereotyped(self.bodyPart, sequenceNode) inv: self.result->isEmpty() inv: self.loopVariableInput->isEmpty()	
--	--

Textual notation

```

«while» ::= <while header> '{' <while body> '}'
<while header> ::= 'while' '(' [ <while condition> ] ')'
<while condition> ::= <Boolean expression>
<while body> ::= <statements>

```

Semantics

«while» is a construct to provide iterations of activities with the help of a condition. The condition of the execution is evaluated before any loop body is executed. The setupPart defines the initialization part which is executed before any run of the inner loop. It is only done once. The bodyPart defines the inner loop behavior. The test and decider attributes define the exit condition and decide on the abortion of the while loop.

7.4.15 Repeat

The LoopNode is extended with the stereotype «for» from the metaclass LOOPNODE (FROM COMPLETESTRUCTUREDACTIVITIES, STRUCTUREDACTIVITIES).

UML NODE TYPE	UML NOTATION	REFERENCE
LoopNode	- No graphical notation -	12.3.35 LoopNode
UML CS STEREOTYPE	UML METACLASS	
«repeat»	LoopNode (from CompleteStructuredActivites, StructuredActivities)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context LoopNode inv: self.isTestedFirst=false inv: self.setupPart->size()<=1 inv: self.setupPart->size()=1 implies isStereotyped(self.setupPart, sequenceNode) inv: self.test->size()<=1 inv: self.test->size()=1 implies isStereotyped(self.test, sequenceNode) inv: self.bodyPart->size()=1 inv: isStereotyped(self.bodyPart, sequenceNode) inv: self.result->isEmpty() inv: self.loopVariableInput->isEmpty()		

Textual notation

```

«repeat» ::= <repeat header> '{' <repeat body> '}' <repeat trailer>
<repeat header> ::= 'do'
<repeat body> ::= <statements>
<repeat trailer> ::= 'while' '(' <repeat condition> ')' <semicolon>
<repeat condition> ::= <Boolean expression>

```

Semantics

«repeat» is a construct to provide iterations of activities with the help of a condition. The repeat-body is executed before the repeat condition is evaluated. Therefore, the condition of the execution is evaluated after the repeat body is executed. The execution of the repeat-body is repeated as long as the repeat-condition yields a true value. This implies that the repeat-body is executed at least once. The setupPart defines the initialization part which is executed before any run of the inner loop. It is only done once. The bodyPart defines the inner loop behavior. The test and decider attributes define the exit condition and decide on the abortion of the loop.

7.4.16 No Operation

The OpaqueAction is extended with the stereotype «noOperation» from the metaclass OPAQUEACTION (FROM BASICACTIONS). The «noOperation» action does neither have any effects nor changes any features or states.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
OpaqueAction	- No graphical notation -	11.3.26 OpaqueAction
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«noOperation»	OpaqueAction (from BasicActions)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context OpaqueAction inv: self.body->size()=0 inv: self.language->size()=0		

Textual notation

```

«noOperation» ::= <semicolon>


```

Semantics

This «noOperation» element does not have any effect on the current activity. It does not modify any structural or behavioral classifier. This statement can be used if statements have to be unspecified. For instance, the initialization part of a For-loop if nothing has to be declared or initialized prior to the execution of the For-loop body.

7.4.17 Begin

The InitialNode is extended with the stereotype «begin» from the metaclass INITIALNODE (FROM BASICACTIVITES). An initial node is a control node at which flow starts when the activity is invoked.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
InitialNode		12.3.31 InitialNode
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«begin»	InitialNode (from BasicActivities)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context InitialNode inv: self.outgoing->size()=1 inv: self.incoming->size()=0 inv: self.inGroup.nodes->select(n isStereotyped(n,begin))->size()<=1		There shall be at most one begin node within one activity.

Textual Notation


```
«begin» ::= 'begin' «sequenceNode» 'end'
```

Semantics

The «begin» stereotype node is a starting point for executing an activity. Note that a control flow can also start at ActivityParameterNode which receives the parameters passed to an activity. Therefore, begin nodes are not mandatory for an activity to start execution.

7.4.18 Return

The ActivityFinalNode is extended with the stereotype «return» from the metaclass ACTIVITYFINALNODE (FROM BASICACTIVITES, INTERMEDIATEACTIVITES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
ActivityFinalNode		12.3.6 ActivityFinalNode
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	<i>TAGEED VALUES</i>
«return»	ActivityFinalNode (from BasicActivities, IntermediateActivites)	argument: Property[0..1]
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>

context ActivityFinalNode inv: self.argument->oclIsKindOf(ValuePin)	
--	--

Textual Notation

<pre>«return» ::= 'return' [<argument>] <semicolon> <argument> ::= <expression></pre>

Semantics

The «return» node is a final node and terminates the activity. It stops all executing actions in the activity with the exception of output activity parameter nodes. By the termination of the execution of the synchronous invocation actions, also the behaviors that are waiting for in order to return are terminated. If there is more than one final node in an activity and the first one is reached, it terminates the activity. Any object nodes declared as outputs are assigned their respective values. All object nodes declared as outputs or return values are passed out of the containing activity. If they are empty, the null token which stands for object nodes that have nothing in them is returned. If the return node has an argument assigned, this value is returned to the caller as the activities result.

7.4.19 Continue

The OpaqueAction is extended with the stereotype «continue» from the metaclass OPAQUEACTION (FROM BASICACTIONS). The continue instruction causes the current execution to skip the rest of the loop in the present iteration and immediately continuing the execution with the subsequent iteration.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
OpaqueAction	- No graphical notation -	11.3.26 OpaqueAction
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«continue»	OpaqueAction (from BasicActions)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context OpaqueAction inv: self.inputValue->isEmpty() inv: self.inGroup-> forAll(g g->oclIsKindOf(LoopNode))		A continue node shall only be used within a LoopNode.

Textual notation

«continue» ::= 'continue' <semicolon>

Semantics

If the «continue» node is executed, the current execution of nodes is skipped and the execution is continued with the condition evaluation of the containing loop. This node can only be placed within a conditional or unconditional loop.

7.4.20 Break

The OpaqueAction is extended with the stereotype «break» from the metaclass OPAQUEACTION (FROM BASICACTIONS). A break statement can be used to leave a loop even if the condition for its end is not fulfilled. A common application is to prematurely end an infinite loop or end a conditional loop before its natural end.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
OpaqueAction	- No graphical notation -	11.3.26 OpaqueAction
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«break»	OpaqueAction (from BasicActions)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context OpaqueAction inv: self.inputValue->isEmpty() inv: self.inGroup-> forall(g g->oclIsKindOf(LoopNode))		A break node shall only be used within a LoopNode.

Textual notation

```
«break» ::= 'break' <semicolon>
```

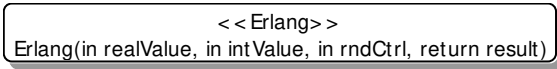
Semantics

When a «break» element is reached, the current execution immediately exits its innermost surrounding loop node and the execution is continued at the next action following the loop node. This node can only be placed within a conditional or un-conditional loop.

7.5 Random

Especially for performance and robustness analysis and simulation of communication protocols, an important feature is the possibility to generate random numbers. Non-determinism is only in very rare cases efficiently suited to yield random values that are required to model packet losses for packet switched networks, especially in wireless environments. Therefore, well-designed random generators are required with a sufficient number of different distributions.

Furthermore, it is also argued that the generated random number sequences are re-producible. This is required to be able to run a slightly modified version of a system with the same sequence of generated random numbers. A random generates a pseudo random value which is derived from a given random seed value. The sequence of random value numbers is identical in each run of the system. This is especially required for high amount of participating entities and high amount of communication. In particular, when an analysis of a system with a random value shows a specific (erroneous) behavior, a second run with a different set of random values might result in correct behavior. In this case, it would be hard for the engineer to re-produce the random values that led to the failure. In addition, automatic validation of specification would introduce an additional high amount of complexity. Therefore, varying random values sequences can only be generated by means of different random seed values.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
CallOperationAction		11.3.10 CallOperationAction
<i>UML CS STEREOTYPE</i>	<i>UML CS METACLASS</i>	<i>TAGGED VALUES</i>
«random»	CallOperationAction (from BasicActions)	arguments: Property [0..*] isAbstract=true
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context CallOperationAction inv: self.operation.method-> isEmpty() inv: self.onPort->isEmpty()		There shall be no behavioral implementation of the operation's <u>method</u> as this is internally done. The CallOperationAction shall not denote an <u>onPort</u> rendering it a remote call. This stereotype is abstract. It is specialized by the noted random classes.

Semantics

When a «random» is executed it returns a pseudo-random value. Inspired from the Tau G1 [Tau] random library, following operations are provided for randomness. Note that the following random operations are all specialized forms of the random stereotype noted above. The following notation is used as shorthand for replacing the stereotype tables:

```
RandomControl: Integer -> RandomControl;
```

RandomControl creates a *RandomControl* instance with an initial seed value. The seed value is used to generate the random values. Identical seed values yield the same sequence of random values.

```
Random : RandomControl -> Real;
Random : RandomControl -> Duration;
Random : RandomControl -> Time;
```

Random generates a random value x , where $0.0 \leq x \leq 1.0$.

```
Erlang : Real, Integer, RandomControl -> Real;
Erlang : Real, Integer, RandomControl -> Duration;
Erlang : Real, Integer, RandomControl -> Time;
Erlang : Duration, Integer, RandomControl -> Real;
Erlang : Duration, Integer, RandomControl -> Duration;
Erlang : Duration, Integer, RandomControl -> Time;
Erlang : Time, Integer, RandomControl -> Real;
Erlang : Time, Integer, RandomControl -> Duration;
Erlang : Time, Integer, RandomControl -> Time;
```

Erlang creates an Erlang-N distributed random value. The first value specifies the mean, the second value specifies N.

```
NegExp : Real, RandomControl -> Real;
NegExp : Real, RandomControl -> Duration;
NegExp : Real, RandomControl -> Time;
NegExp : Duration, RandomControl -> Real;
NegExp : Duration, RandomControl -> Duration;
NegExp : Duration, RandomControl -> Time;
NegExp : Time, RandomControl -> Real;
NegExp : Time, RandomControl -> Duration;
NegExp : Time, RandomControl -> Time;
```

NegExp creates a negative exponential distributed random value with the specified mean.

```
Uniform : Real, Real, RandomControl -> Real;
Uniform : Real, Real, RandomControl -> Duration;
Uniform : Real, Real, RandomControl -> Time;
Uniform : Duration, Duration, RandomControl -> Real;
Uniform : Duration, Duration, RandomControl -> Duration;
Uniform : Duration, Duration, RandomControl -> Time;
Uniform : Time, Time, RandomControl -> Real;
Uniform : Time, Time, RandomControl -> Duration;
Uniform : Time, Time, RandomControl -> Time;
```

Uniform creates a random value that is uniformly distributed between the ranges that are specified by the two values. The range is specified including the values.

```
Draw : Real, RandomControl -> Boolean;
```

Draw performs a single randomized test for the specified probability. That is, the operator *Draw* yields a true value with the given probability x and a false value with the probability $1-x$.

```
geometric : Real, RandomControl -> Integer;  
geometric : Real, RandomControl -> Duration;  
geometric : Real, RandomControl -> Time;  
geometric : Duration, RandomControl -> Integer;  
geometric : Duration, RandomControl -> Duration;  
geometric : Duration, RandomControl -> Time;  
geometric : Time, RandomControl -> Integer;  
geometric : Time, RandomControl -> Duration;  
geometric : Time, RandomControl -> Time;
```

Geometric returns an Integer value based on geometric distribution.

```
Poisson : Real, RandomControl -> Integer;  
Poisson : Real, RandomControl -> Duration;  
Poisson : Real, RandomControl -> Time;  
Poisson : Duration, RandomControl -> Integer;  
Poisson : Duration, RandomControl -> Duration;  
Poisson : Duration, RandomControl -> Time;  
Poisson : Time, RandomControl -> Integer;  
Poisson : Time, RandomControl -> Duration;  
Poisson : Time, RandomControl -> Time;
```

Poisson returns a random value based on the Poisson distribution with the first value specifying the mean value.

```
RandInt : Integer, Integer, RandomControl -> Integer;  
RandInt : Integer, Integer, RandomControl -> Duration;  
RandInt : Integer, Integer, RandomControl -> Time;
```

RandInt returns an Integer value between the specified ranges. The values are equally distributed.

The following Figure 72 gives an overview of the random passive classes which are available in this profile. The *CallOperationAction* with the random extension invokes the corresponding operation implementation of the specific class. These classes are passive classes and do not execute a behavior unless an operation is called.

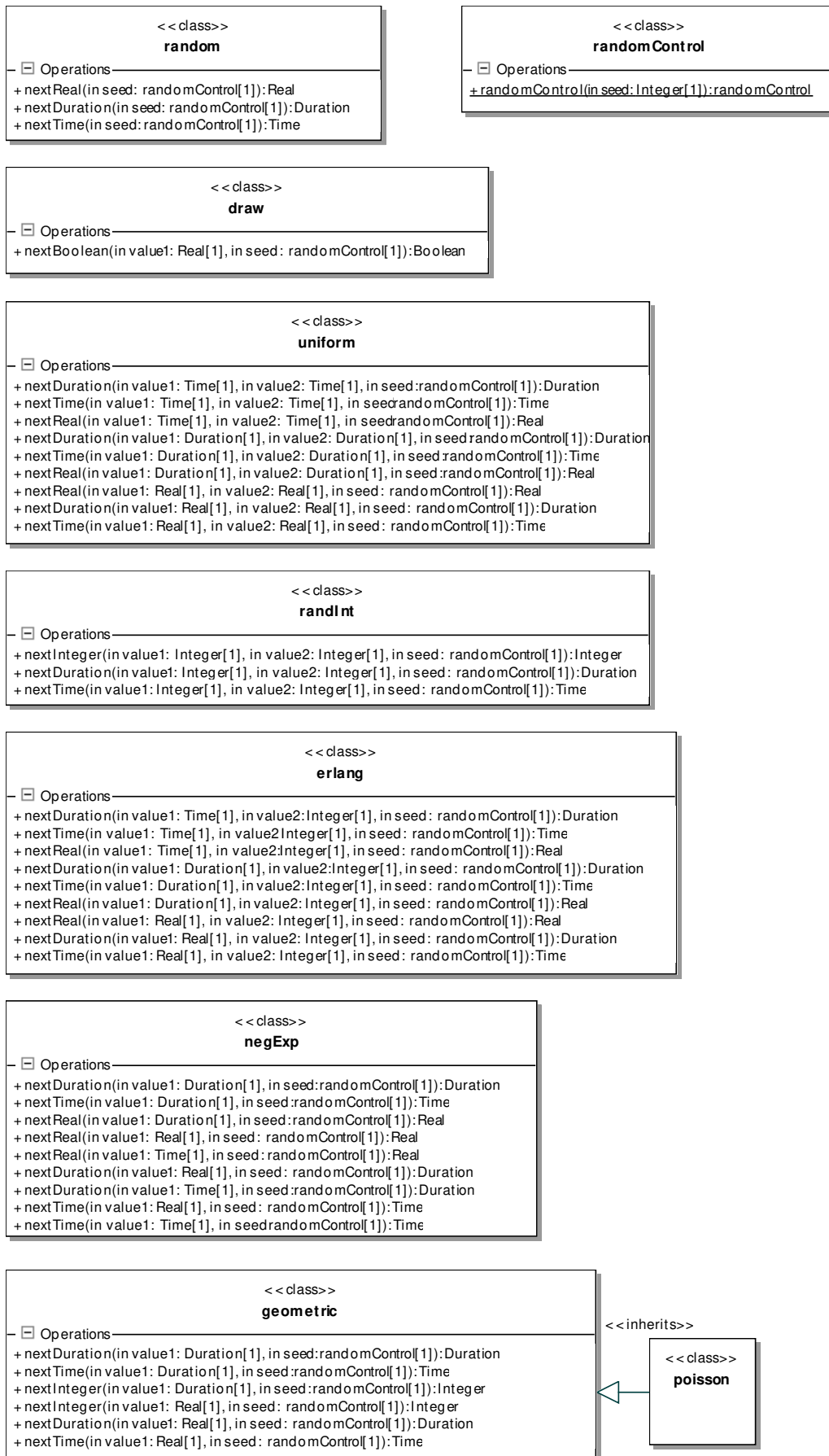


Figure 72: Predefined Set of Random Classes

7.6 Concept of Time

When specifying communication protocols, time concepts are essential in order to provide precise and complete specifications. Timers are used as a means to manipulate and control a communication protocol's behavior and to verify time bounds and proper termination. The UML 2 features the *SimpleTime* concept but this does not cover the full needs for communication protocol specifications. The *SimpleTime* package allows to start a timer implicitly and to wait until its timeout. This is accomplished using the expressions *after* for relative time bounds and *at* for absolute time. The unit for time value is usually specified informally.

For communication protocols it is necessary to start a timer independently from waiting on the timeout trigger. The timeout of a timer must be able to occur after several actions have been taken. The UML 2 profiles specified in [OMG05b, OMG05c] also introduce an own time concept which has a greater applicability than the basic UML 2 time concept. The [OMG05c] specifies a time concept for real time purposes, including features like skew, stability, drift, time zones and density. As its usage introduces high complexity and high requirements to simulators, validators and code generators a less exact time concept is used in this profile. It is inspired from the UML 2 testing profile specified in [OMG05b].

The following metaclasses of the UML are extended in this section: Signal, WriteStructuralFeatureAction and ReadStructuralFeatureAction.

Timer and Time Trigger

A *timer* is a pre-defined class and active objects can own the instances. An active class may own multiple instances. Pre-defined operations like *set*, *reset* and *read* are defined for the timer objects. For example, "*set(timer1, now+2)*" starts the *timer* instance *timer1* and sets its expiration time to the current system time increased by two time units. When the system timer reaches this timeout value from the *timer1*, *timer1* triggers a *TimeEvent*. With the *reset()* operation a timer can be stopped. When a timer is stopped a *TimeEvent* cannot be triggered. The *active()* operation allows to read the time when the timer is scheduled to timeout. It results a true Boolean value if the timer is currently scheduled to trigger a timeout signal. Following Figure 73 gives an overview of the extended metaclasses. A timer is a specialization of a signal with an association to the *TimeExpression* metaclass. The *TimeExpression* defines the timeout by a *ValueSpecification*. The timer's attributes can be read and written by the appropriate *StructuralFeatureActions*.

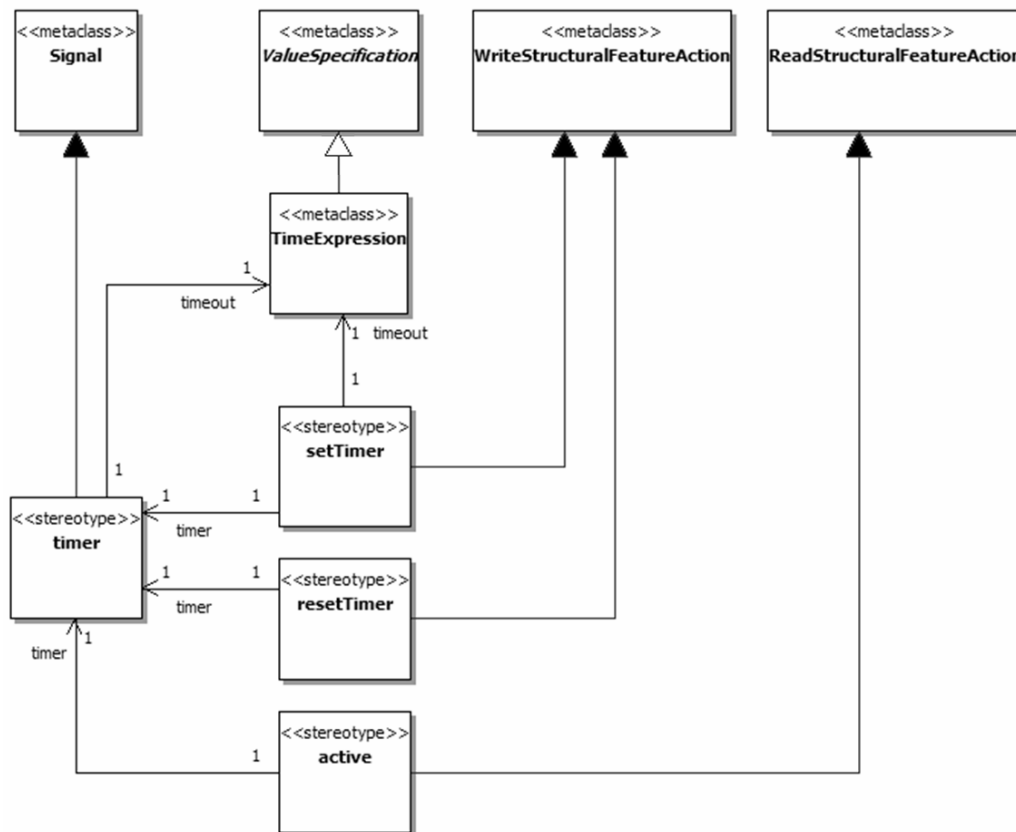



Figure 73: UML CS Profile Timer Concept

The recommended graphical notation for all timer related actions is an empty hourglass as shown in the following Figure 74, Figure 75 and Figure 76. The hourglass has one single incoming transition and one single outgoing transition. Alternatively, a textual notation is provided, e.g. in a task box.

7.6.1 Timer

The reception of a Timer is an event that can trigger transitions in the same fashion as a signal. A timer is initialized by a state machine. When a timeout is reached, the owning state machine can receive a timer signal. A time value is associated with an active timer which is the point of timeout.

UML NODE TYPE	UML NOTATION	REFERENCE
Signal		13.3.23 Signal
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
<<timer>>	Signal (from Communications)	timeout: Time
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context Signal inv: self.name->notEmpty()		

```
inv: self.ownedOperation->isEmpty()
```

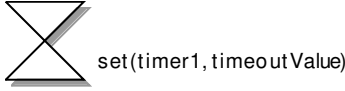
Semantics

Similar to signals, a «timer» may have parameters. Therefore, this does allow setting of more than one timer of the same kind. It is not required to reset an already active timer. That is, several timers with different parameter values may be active at the same time.

The ownedAttribute attribute defines the parameters of the timer instance. The timeout tag definition defines the time when a timeout shall occur.

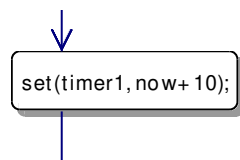
7.6.2 Timer Start

WriteStructuralFeatureAction is extended with the stereotype «setTimer» from the metaclass WRITESTRUCTURALFEATUREACTION (FROM STRUCTURED ACTIONS). «setTimer» is an action to start a timer with a defined value when a timeout occurs. A timeout does not occur if the given time is lower than current system time or if the timer is not running.

UML NODE TYPE	UML NOTATION	REFERENCE
WriteStructuralFeatureAction		11.3.54 WriteStructuralFeatureAction
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«setTimer»	WriteStructuralFeatureAction (from StructuredActions)	timer: Timer timeout: TimeExpression
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
		<u>timer</u> and <u>timeout</u> are derived values from the <u>StructuralFeature</u> and the from the <u>value</u> property.

Textual Notation

```
<setTimer> ::= 'set(' <timer identifier> ',' <time expression> ')'  
<semicolon>
```



(a) Textual notation in a Task Box



(b) Timer Expression

Figure 74: Notations for Starting Timer

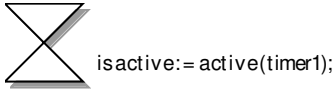
Semantics

A process can own timers. A timer has a defined timeout expression that – when being equal to the current system time (*now*) – will trigger a timer signal to be sent to its owning process. The timer signal will be sent only once. The time when the timer fires a timer signal is specified in the timeout attribute. For relative time specification, the pre-defined expression *now* is available representing the current system time when execution is running.

The timer and timeout tag definitions define the timer and the value when a timeout shall occur.

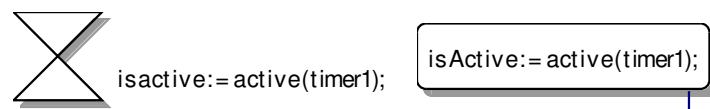
7.6.3 Timer Status

The ReadStructuralFeatureAction is extended with the stereotype «timerActive» from the metaclass READSTRUCTURALFEATUREACTION (FROM STRUCTURED ACTIONS). When being executed «timerActive» is an action that obtains the status of timeout time of the specified timer when the timeout occurs. If the result is that the timer is currently active and a timeout is scheduled to occur, it writes a true value to the given structural feature. If one of the conditions is not satisfied, it writes a false Boolean value.

UML NODE TYPE	UML NOTATION	REFERENCE
ReadStructuralFeatureAction		11.3.36 ReadStructuralFeatureAction
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«timerActive»	ReadStructuralFeatureAction (from StructuredActions)	timer: Timer
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
		The <u>timer</u> tagged value is derived from the <u>StructuralFeature</u> property.

Textual Notation

```
«active» ::= 'active(' <timer identifier> ')' <semicolon>
```



(a) Timer Expression (b) Textual notation in a Task Box

Figure 75: Notations for Reading Timer's Status


Semantics

A timer can be tested if it is active. Active means that a timer has been started and that the timer event has not yet occurred. In addition, its timeout expression is greater or equal than the current time (*now*).

If the timer object is active, the stereotype reads a true value, otherwise a false value. The timer tag definition defines the timer that is to be read. The result attribute defines the target variable or pin.

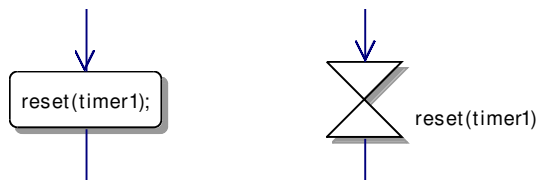
7.6.4 Timer Reset

The WriteStructuralFeatureAction is extended with the stereotype «resetTimer» from the metaclass WRITESTRUCTURALFEATUREACTION (FROM STRUCTURED ACTIONS). «resetTimer» is an action to stop a running timer. If the timer is not running, it does nothing.

UML NODE TYPE	UML NOTATION	REFERENCE
WriteStructuralFeatureAction		11.3.26 WriteStructuralFeatureAction
UML CS STEREOTYPE	UML METACLASS	TAGGED VALUES
«resetTimer»	WriteStructuralFeatureAction (from StructuredActions)	timer: Timer
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
		The <u>timer</u> tagged value is derived from the <u>StructuralFeature</u> property.

Textual Notation

```
«resetTimer» ::= 'reset(' <timer identifier> ')' <semicolon>
```



(a) Textual notation in a Task Box (b) Timer Expression

Figure 76: Notation for Resetting Timer

Semantics

A timer can be stopped by re-setting it. If a timer is stopped, a time event caused by this timer does not occur until it is re-started. The timer tag definition defines the timer that is to be re-setted.

7.7 Data types

The data types and composite data types, which are available in the profile described in this thesis, are derived from the SDL-2000 data type concept. A data type is a label or name for a set of values including operations which can be performed on these values. In SDL-2000, following constructs are available to define data [Loe03]:

<pre><data definition> = <data type definition> <interface definition> <syntype definition> <synonym definition></pre>
--

In this definition, only the <data type definition> is used to define new data type. An interface contains the definition of signals, signatures of procedures and remote variables and determines the possibilities of interactions of an agent. Values of an interface realize references to agents. A syntype defines an alias name for an existing data type and can constrain the value domain of the data type. A synonym defines a symbolic constant. As a new feature of SDL-2000, the data type definition is possible for object (reference) types and for value types. While an assignment of a value type results in a copy of the concrete values, an object type assignment based on reference types is only a copy of the reference.

Data types are defined by means of abstract data types. SDL is based on the concept that is equivalent to the ACT ONE [EM85] and LOTOS notation. In addition to the Z.100 standard, a detailed introduction to the data type concept can be found in [EHS97, Sch03a, Loe03].

The following metaclasses of the UML are extended in this section: PrimitiveType, DataType, Enumeration, Generalization and Property.

7.7.1 Abstract Data Types

The aim of abstract data types (ADT) is the description of data structures independent from the implementation and keeping an appropriate level of abstraction. There are two main principles that can be derived from the concept of ADTs:

- Encapsulation: an ADT can only be accessed by its available interfaces. The interface is provided by the operations definition. This is the concept of the abstraction.
- Obscurity: the realization, which is the platform-specific implementation of the ADT, is hidden because an ADT is independent from the concrete representation. Accessing the data type can therefore only be executed by the set of operations which are provided.

The advantages of ADTs are the provision of consistency and flexibility. The consistency results from the fact that an object cannot be set into an invalid condition, as it is only accessible through its provided operations. The flexibility arises from the ease of changing the implementation details after the definition. This makes the difference to concrete data types which always refer to a base data type definition in a specific environment, for instance C++. So they can only be used inside this environment.

An abstract data type is a structure of domain values, operations defined on this basis, and a set of axioms and preconditions. The specification of data types is separated into two parts. First, the syntax of the data type is defined by its *signature*. Second, the semantics is specified which gives a concrete domain and operations to the signature.

A signature contains a set of *sorts* and a set of *operations*. A sort represents a name for the object set while operations represent a name for operations that perform on basis of these sets. For instance, a definition of an Integer data type may look like the following:

```
datatype Integer
  Sort          int
  Operations    empty:      → int
               +: int, int → int
               -: int, int → int
enddatatype
```

This specifies the signature of the data type Integer. The name of the sort is *int*. There are no details concerning the meaning of *int*. Besides of the sort, there are three operations defined. First there is the zero-ary empty operation that is used to create an element of the sort *int*. Second, the operations “+” and “-” are defined which are two-ary functions. This defines that these function map a pair of elements of *int* sorts to one single element of an *int* sort. How this is actually achieved, is not specified, even one might assume that the “+” operations perform an arithmetic addition on an integer value. This is because it is not known what is represented by the sort *int*.

This is part of the semantic description of an abstract data type. The semantic description defines a concrete domain for the sort, for instance the domain of natural numbers, a set of axioms and a set of preconditions that constrain the possible input values for the operations to work correctly. For example, some axioms for the “+” operations could be

```
+ (x, y) = x + y
- (x, y) = x - y
```

As already noted, the SDL definition of ADTs is based on the algebraic specification language ACT ONE. In this language, data specifications are collected into type constructions. A type consists of a set of sorts that represents the possible sets of values, a set of operations describing the signature of the type functions and a set of equations written as equalities of expressions of the type.

An example for definition of an ADT *Boolean* in SDL-96 syntax is the following:

```
newtype Boolean
  literals True, False;
  operators
  "and" : Boolean, Boolean -> Boolean;
  "not" : Boolean -> Boolean;
  axioms
  "and"(True, True) = True;
  "and"(True, False) = False;
  "and"(False, True) = False;
  "and"(False, False) = False;
  "not"(True) = False;
  "not"(False) = True;
endnewtype;
```

This notation specifies the sort Boolean with the literals True and False. There are two operations available with “and” being a two-ary operation mapping two Boolean values to a single Boolean value. And the one-ary “not” operation which maps a single Boolean value to a single Boolean value. The axioms for this ADT are given for both operations. Both operations are defined according to the known Boolean logic function *and* and *not*.

With the introduction of SDL-2000 this type of data type definition has been replaced by an algorithmic definition. In addition to the value data types, object types with reference semantics are introduced.

Figure 77 shows the metamodel of the various data types including the value and object type associations with struct and union data type which will be described in the following.

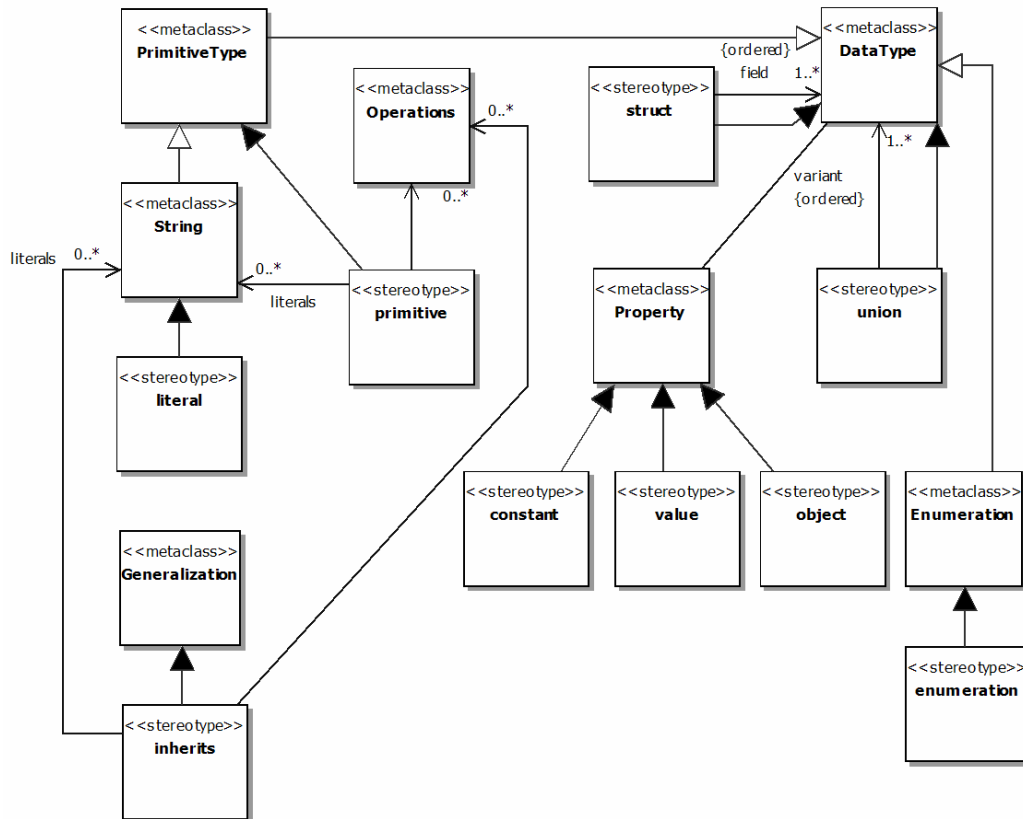


Figure 77: Data Type Metamodel

7.7.2 Primitive and Composite Data Types

7.7.2.1 PRIMITIVE DATA TYPES

Similar to SDL-2000 where the term axiom has been replaced by method the UML CS profile provides three parts for the complete specification and description of an ADT:

- Literals specified as class attribute of a set of Strings
- Operations which specify the operators of the ADT
- Methods which specify the axioms of the ADT

Operations (Operators) have to be specified by algorithmic definitions and are allowed to have side effects. For this purpose, the «primitive» stereotype is used which defines an abstract data type. The «primitive» stereotype extends the UML metaclass PRIMITIVE_TYPE (FROM KERNEL) with the attribute *constructor* that is provided to capture the literals of the ADT. Alternatively, this can be used for composite data types. A primitive type defines a pre-defined data type and does not have any relevant substructure. A primitive (or abstract) data type has an algebra algorithmically defined which is done by an activity.

UML NODE TYPE	UML NOTATION	REFERENCE
PrimitiveType	<pre> <<primitive>> Boolean - Attributes + literals[2]={"True", "False"} - Operations + and(in x: Boolean[1], in y: Boolean[1]): Boolean + not(in x: Boolean[1]): Boolean </pre>	7.3.43 PrimitiveType
UML CS STEREO TYPE	UML METACLASS	TAGGED VALUE
<<primitive>>	PrimitiveType (from Kernel)	literals: String[0..*]
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
<p>context Primitive</p> <p>inv: self.parents()->isEmpty() implies (self.operators->size())>=1 and self.constructor->notEmpty()</p> <p>inv: self.ownedOperation-> forAll(not visibility=VisibilityKind::package)</p> <p>inv: self.ownedOperation->forAll(o o.isStatic)</p> <p>inv: self.ownedOperation-> forAll(o o.method->oclIsKindOf(Activity))</p> <p>inv: self.extension_primitive.literals->size()=0 implies self.general->size()=1</p>		<p>If this <i>primitive</i> is not inherited (generalized), there shall be at least one literal defined.</p> <p>The visibility of the operations is limited to public, protected and private (not package). All operations shall be static (that means, the operations are part of the class and not of the instance. They can be invoked by the class reference). The behavior of an operation must be specified by an activity only (not by a state machine).</p>

7.7.2.2 INFIX OPERATORS

For infix operators, the operation's name must be noted like "=", ">", "<". For example, refer to the following Figure 78:

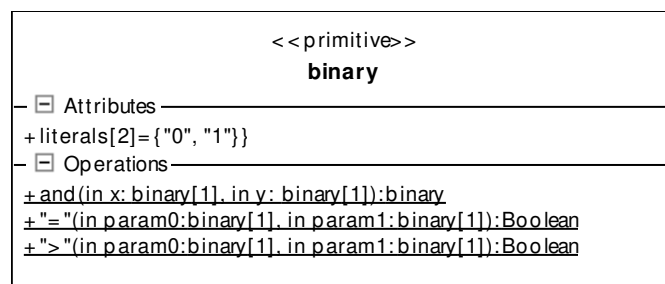


Figure 78: Primitive Definition with Infix Operations

The definition of a primitive type *binary* is shown with its literals "0" and "1" and with three operations: *and*, "=", ">". The quotation marks denote infix notation. Therefore, the operation

```
"=" (in param0: binary[1], in param1: binary[1]):Boolean
```

is invoked if an expression like

```
0=1
```

is executed. This “=” operation takes two parameters of binary type and returns a Boolean value (if both binary type variables are equal or not).

7.7.2.3 SEMANTICS

At run-time, the instances of a primitive type are mapped to data values and have no identity. The visibility of operations introduce the following restrictions: A *private* visibility of an operation implies that the operation can only be accessed and resolved by the value type definition in which it is defined. Specialized types cannot access these operations. *Protected* operations imply that the operation is accessible from the specialized subtypes, but not from external. *Public* visibility does not limit any access rule. Different from language like C++ private and protected operations are not available if an operation name resolution is performed. In C++ private and protected operations can be resolved, but their invocation will result in an error.

For the resolution which operators are to be invoked, the *resolution by context* scheme is applied which is identical to SDL-2000. For example, for the expression 2+3 it is not clear if the operation “+” of the primitive type Integer, Natural or Real is to be called. This scheme is described in [Loe03] and is cited here in the following:

- If a data type is visible at a specific location within the specification, all literals and operations of the data type are visible.
- For actions like assignment, timer start or a signal sending, all operations that belong to this action are identified and for each of these operation all visible definitions.
- For each of the definition is checked if it is type compatible. If all definition fail to comply, the expression is considered erroneous.
- From the type compatible set of operations, this one is selected which has the lowest amount of polymorphic type inheritances used.
- If there are multiple operations available for selection, the expression is ambiguous and therefore considered erroneous. Is there exactly one distinct interpretation, the expression is correct.

7.7.2.4 PRE-DEFINED TYPES

SDL provides the following pre-defined types which are also available in this UML CS profile (for a detailed list of defined literals and operations see Annex D of the Z. 100 standard. In addition, examples for possible values are provided):

- Boolean: True, False
- Character: ‘A’, ‘c’, ‘7’, ...
- Charstring: ‘This is a charstring’
- String: a generic string (defines a list of elements of any type, not limited to Characters)

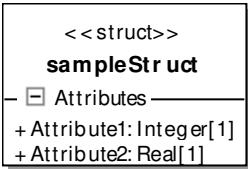
- Integer: -4, 0, 7888, ...
- Natural: 0, 5, 88, 5869, (non negative Integers)
- Real: -3577.7, 68.8, ...
- Pid: a unique identification for processes (a reference to agents)
- Duration: a relative time value, e.g. difference of two Time values used for timers
- Time: an absolute value of time, used for timers
- Bit: 0,1
- Bitstring: a string of type Bit
- Octet: '010110101' (a Bitstring with a length of eight)
- Octetstring: a string of type Octet
- Array: a generic array
- Powerset: a generic set
- struct: a composition of several data types (a structured type)
- choice: an exclusive alternative of data types (a union or choice type)

For all types, the operations “:=”, “=” and “/=” are implicitly pre-defined. The operation “:=” is an assignment of the resulting value expression calculation on the right side to the variable on the left side. The operation “=” is a two-ary operation which yields a true Boolean value if both type values are equal otherwise false. The operation “/=” is a two-ary operations which yields a true Boolean value if both type values are not equal otherwise false. In detail, all types have the following operators implicitly defined:

“:=”:	value, value -> Boolean
“/=”:	value, value -> Boolean

7.7.2.5 STRUCT

A data type in SDL-2000 can be defined by means of constructors. Constructors available are the literal, struct and union. Only one of these constructors shall be specified for a new type.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
DataType		7.3.11 DataType
<i>UML CS STEREO TYPE</i>	<i>UML METACLASS</i>	<i>ATTRIBUTES</i>
<<struct>>	DataType (from Kernel)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>

context DataType inv: self.ownedAttribute->size()>0	
--	--

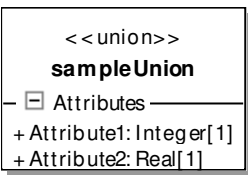
A «struct» composes an ordered set of attributes into a logical grouping. Accessing an attribute of a structure, a *field*, is possible by SDL-2000 syntax using ‘.’ or ‘!’, e.g.:

```
d:=PDUtype.data
d:=PDUtype!data
```

A data type specifies either a structure (struct) or union or primitive type. A struct also defines an ordered set of value types of which it consists.

7.7.2.6 UNION

The same as for the struct type applies for a union data type. However, a union can only specify exactly one of the attributes at the same time. An assignment to a union erases all previous variants that are defined in the union. Accessing an attribute, a *variant*, is possible by the same syntax as for a struct.

UML NODE TYPE	UML NOTATION	REFERENCE
DataType	 <pre> classDiagram class sampleUnion { + Attribute1: Integer[1] + Attribute2: Real[1] } </pre>	7.3.11 DataType
UML CS STEREOTYPE	UML METACLASS	ATTRIBUTES
«union»	DataType (from Kernel)	
OCL CONSTRAINTS		INFORMAL CONSTRAINTS
context DataType inv: self.ownedAttribute->size()>0		

Nested composite data types can also be specified by the recursive declaration of data types and composite types.

7.7.3 Value and Object types

The profile categorizes two different types of data: value types and object types. The difference between these two types lies in the assignment semantics. While a value type assignment is an actual copy of the whole data type instance, the object type assignment is only a copy of the reference. The stereotypes for both types are introduced in the following.

Variables and operations declared in an agent have their scope according to their specified visibility:

- *Public (+)*: the attribute or operation is visible to all other agents.

- *Private (-)*: the attribute or operation is visible only to its owning agent.
- *Protected (#)*: the attribute or operation is visible only to its owning agent and to all agents that are a specialized form of the owning agent.
- *Package (~)*: the attribute or operation is visible only to agents which are defined in the same package.

The term *visibility* includes that neither the attribute nor the operation can be read, written, accessed, referenced or called by an agent that is not included in the visibility scope.

7.7.3.1 VALUE TYPE

The «value» stereotype (valueType) extends the UML metaclass PROPERTY (FROM KERNEL, ASSOCIATIONCLASSES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Property	- no specific notation -	7.3.44 Property
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«value»	Property (from Kernel, AssociationClasses)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Property inv: self.isAttribute()		

Semantics

A «value» property defines value-type semantics to property of an agent. This property has the characteristic that assignments to variables result in the copy of the data type value. This allows the assignment of concrete values to this type.

7.7.3.2 OBJECT TYPE

The «object» stereotype (objectType) extends the UML metaclass PROPERTY (FROM KERNEL, ASSOCIATIONCLASSES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Property	- no specific notation -	7.3.44 Property
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«object»	Property (from Kernel, AssociationClasses)	

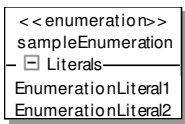
<i>OCL CONSTRAINTS</i>	<i>INFORMAL CONSTRAINTS</i>
context Property inv: self.isAttribute()	

Semantics

The semantics of an «object» is similar to the «value». The difference is the reference semantics for instances of an object. An assignment to an instance of an object is the assignment of a reference (a pointer) of the assigned instance (reference semantics). A concrete value cannot be assigned to a property with object-type semantics.

7.7.4 Enumeration

An enumeration is extended by «enumeration» from the metaclass ENUMERATION (FROM KERNEL). An enumeration is a data type whose values are listed in the model as enumeration literals which are represented by distinct names.

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Enumeration		7.3.16 Enumeration
<i>UML CS STEREOTYPE</i>	<i>UML METACLASS</i>	
«enumeration»	Enumeration (from Kernel)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Enumeration inv: self.interfaceRealization->size()=0 inv: self.general->size()<1		Multiple inheritances are not allowed for an enumeration. An enumeration shall not have an InterfaceRealization

Semantics

At run-time, an «enumeration» is mapped to values. Every single EnumerationLiteral has one distinct, unique value assigned. An enumeration may also define operations in the ownedOperation attribute.

7.7.5 Constants

A property is extended by «constant» from the metaclass PROPERTY (FROM KERNEL, ASSOCIATIONCLASSES). A property that is extended with the stereotype «constant» declares itself being of immutable value.


<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Property	-No specific notation-	7.3.44 Property
<i>UML CS STEREO TYPE</i>	<i>UML META CLASS</i>	
«constant»	Property (from Kernel, AssociationClasses)	
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Property inv: self.isReadOnly=true inv: self.defaultValue->notEmpty() inv: self.isAttribute() inv: self.lower()>0		Constants are always read only and shall define a lower multiplicity of one or higher. The latter implies that constant values cannot be empty or un-defined.

Semantics

A «constant» defines properties with unchangeable features and values. The main purpose is to introduce a more intelligible textual representation for data values. During mapping to a target language, the symbolic representation of the property may be replaced by its actual value.

7.7.6 Specialization of data types

A data type can be specialized with the following stereotype «inherits» which extends the metaclass GENERALIZATION (FROM KERNEL, POWERTYPES).

<i>UML NODE TYPE</i>	<i>UML NOTATION</i>	<i>REFERENCE</i>
Generalization		7.3.20 Generalization
<i>UML CS STEREO TYPE</i>	<i>UML META CLASS</i>	<i>TAGGED VALUES</i>
«inherits»	Generalization (from Kernel, PowerTypes)	literal: String[0..*] operator: Operation[1..*]
<i>OCL CONSTRAINTS</i>		<i>INFORMAL CONSTRAINTS</i>
context Generalization inv: self.generalizationSet->size()=0 inv: self.specific.parents()->size()=1		

Semantics

«inherits» allows specialization for abstract data types. Inherit can be used to create a new type by inheriting information from another type. It is also possible to specify the operators and constructors that will be inherited, if any, and to add new operators and constructors in the new specialized type.

By default, operators can be overwritten if they are not declared as final. However, the type itself cannot be altered by using inheritance. For instance, further components cannot be added to a struct (structure) when the struct is inherited.

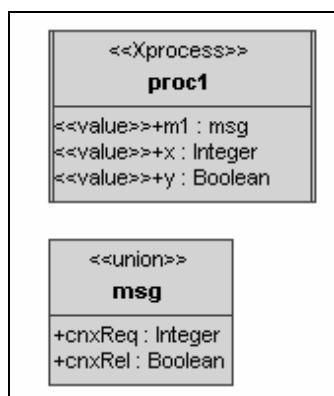
The attribute `literal` is a set of String and specifies the literals to be inherited from a primitiveType. The attribute `operator` is a set of operations and specifies the operators to be inherited. Multiple inheritances are not allowed for data types.

The specialization of data types does not imply that specialized data types are polymorphically usable according to the Liskov substitution principle as remarked in [Loe03], contrary to specialization of agents as described in Section 7.2.8.

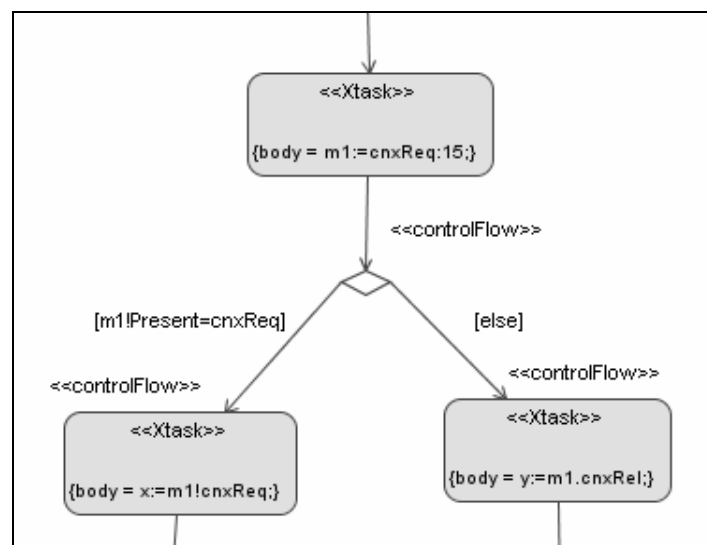
7.7.7 Optional attributes

Optional attributes are an important way to describe communication protocol formats. An attribute can be marked of being *optional*. That is, the attribute may be specified or not, varying from case to case. To indicate optional attributes, the lower multiplicity bound must be set to zero. This indicates that the specific attribute is allowed to remain unassigned.

The presence of an optional field can be evaluated by the suffix *PRESENT* to the field name. This also applies for checking the presence of a union variant. For instance, the following examples (equivalent to the examples given in [Dol01]) in Figure 79 shows the use of the union type with the implicit field present.

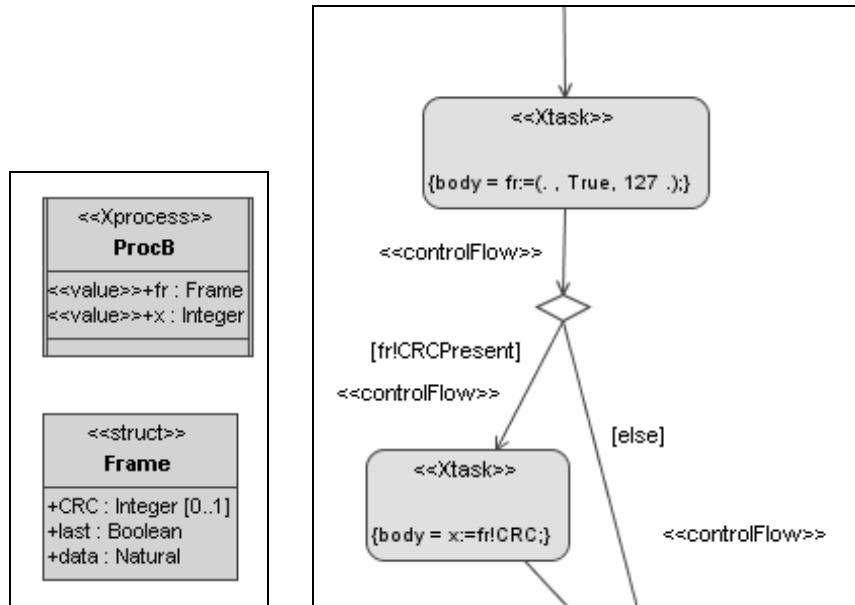


(a) Definition of process and union



(b) Excerpt of Activity testing presence of variant

Figure 79: Union Type and implicit Present Field



(a) Definition of process and struct (b) Excerpt of Activity testing presence of optional field

Figure 80: Struct Type with Optional Field

Figure 80 shows a struct data type with the optional field. This is depicted by the lower multiplicity bound shown on the left process definition. The structure type *Frame* defines the *CRC* field being optional. The presence of this optional field is evaluated by the *fr!CRCPresent* statement. This results in a true value if the *CRC* field is assigned.

7.8 Summary

In this chapter, the UML Profile for Communication Systems (UML CS) has been defined. The profile comprises of a set of semi-formal stereotypes with each of them defining additional semantics and constraints of the extended metaclass. The stereotype constraints have been defined using a formal language, the Object Constraint Language (OCL). The description of the semantics has been done informally. The formal semantics is defined in the next Chapter 8 by means of a formal mapping to Specification and Description Language (SDL-2000).

In the first section, an introduction to the description of a stereotype has been given. In addition, the possibility to define different queue disciplines for processes and their interaction points was explained. This is a specific extension point to enable a future mapping to other formal description techniques such as ESTELLE. The name resolution has also been noted as a pre-requisite because SDL uses a different name resolution as the UML. In the second section, the stereotypes for the structural specification of a system have been defined. This included the hierarchical decomposition capabilities and inter-process communication. The third section has provided the definition of the behavioral stereotypes which were used to define state machines that execute a behavior. Several stereotypes enable the use of many features provided by the UML in this profile as well, such as composite states. The stereotypes have been presented for the definition of a control flow and the modification of objects. In the fourth section, activity stereotypes have been defined. The activity stereotypes provided the means to perform actions, alter the system state or to modify properties and objects. These actions include object creation, operation calls, signal sending, assignments and several conditional loops. The fifth section has defined the random functions which are introduced as a new language feature required for Internet communication protocol modeling and analysis. The sixth

section has given an overview of the stereotypes that introduce the timer mechanisms to this profile. This has included actions for defining and starting a timer as well as means for its configuration. The seventh section has described the data type concepts and has provided a set of stereotypes for their definition. The data type concept was derived from the SDL-2000 data type concept.

8 Semantics of the UML CS Profile

Roughly speaking, formal description languages are based on two parts: Syntax which defines the language constructs that are available and how they can be composed together and semantics which gives a meaning to the language constructs. While the syntax and an informal overview of the UML CS profile semantics have been provided in the Chapter 7, the formal semantics is defined in this chapter. The semantics is described by means of a mapping specification by defining by set of constraints defined in the Object Constraint Language (OCL). The OCL is a formal specification language and the approach of such a formal mapping specification from the UML to SDL is new [WKH06]. The Z.109 standard [ITU99] and its upcoming revision in [ITU06a] provide only an informal mapping.

Some fundamentals to enable this mapping are outlined in the following Section 8.1. The mapping specification constraints are described in the subsequent Section 8.2. UML models, which apply the UML CS profile, can be mapped to a complete structural and behavioral SDL system specification. A proof-of-concept implementation has been created on the basis of the profile definition to validate the feasibility and the soundness of the profile. This implementation by means of an eXtensible Stylesheet Transformation Language (XSLT)-based approach is explained in Section 8.3.

8.1 Translational Semantics for UML CS Profile

It is assumed that a UML model is actually mapped to SDL. During the SDL-2000 compilation process, an SDL specification is successively transformed into an abstract syntax tree, namely AS1 defined in Z.100 main body. This AS1 is the result of parsing and checking by well-formedness conditions of the SDL program. AS1 abstracts away from additional but non-essential expressions like delimiters, keywords, graphical elements, blank spaces and so on only focusing on the relevant information. Furthermore, complex language constructs are decomposed into core concepts.

To validate that the mapping from UML to SDL is correct, the model defined in the UML repository and the system defined in the AS1 of SDL are compared and cross-checked if they fulfill and match specific properties. This comparison must always show that the given constraints are fulfilled. Otherwise, the mapping is considered invalid. There are two pre-requisites for such a comparison: First, for the comparison of values, both data types must be type-compatible. Therefore, before constraints can be applied between both models the data types have to be aligned. Second, each composite object in both the UML repository and the AS1 tree must be uniquely addressable – in other words, the repository and syntax tree must be navigable. For a specification language, the obvious choice for UML-based models is OCL as it is part of the UML standard. OCL supports navigation of the UML metamodel. Therefore, the abstract grammar is mapped to a MOF-compliant metamodel.

This profile drives the mapping from UML to SDL by means of the formal object constraint language OCL 2.0 [OMG05d]. For this purpose a mapping function *Mapping* is defined:

Mapping: UML → SDL

This mapping functions maps a UML model (e.g. given in XMI representation) to an SDL system specification. It is not described how this can be achieved within an implementation. There are OCL expressions specified that constrain the invariant variables or associations of the UML elements that apply to the stereotyped classes.

To specify the mapping performed by the function *Mapping*, only post-operation conditions of the mapping are given. Pre-operation conditions are already specified through the static (invariant) constraints of the profile's stereotypes. The post-constraints validate whether both composite objects (the object in the UML repository and the object in the AS1) are equivalent after the mapping or not. OCL is a declarative language and cannot alter the system state.

The mapping constraints relate to the abstract syntax (abstract grammar) definition of SDL on the left-hand side and to the UML stereotypes attributes and associations on the right-hand side. Notice, the OCL constraints only apply if a UML model element is mapped that has the correct stereotype applied. All constraints given must be preceded by an implication expression as the constraint is always defined in the context of the *Mapping* function. For instance, the correct OCL constraint expression for the name attribute mapping for the state class stereotyped with «state» is the following:

```
context Mapping(sdl: SDL-specification, uml: Classifier, co: Object, e:
Element)
post: isStereotypedBy(e, state) implies co.name=e.name
```

where *sdl* is the root object of the SDL specification, *uml* is the system or package class object, *co* is the composite object of the SDL AS1 tree that is to be validated against the Element *e* defined within the *uml* Classifier. As this applies to all following mapping rules, the context and postconstraint part has been omitted in all constraints.

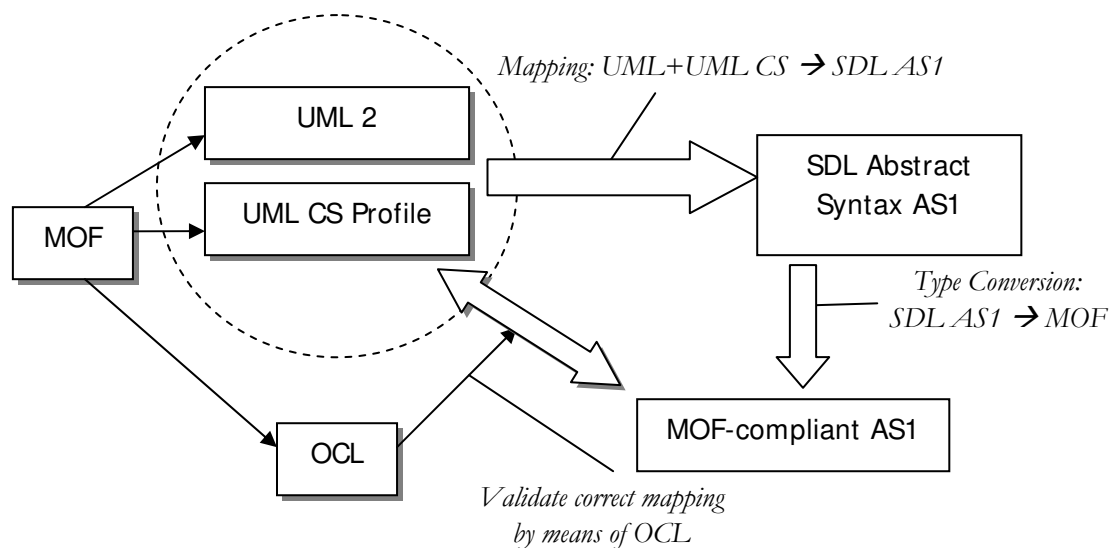


Figure 81: Mapping Specification by OCL

Figure 81 pictorially represents the procedure of the translational semantic mapping to SDL. The mapping is accomplished by the rule that UML model elements with stereotyped extension of the UML CS profile are mapped to the abstract syntax of SDL-2000. How this is finally achieved in an implementation is not specified. However, the following OCL constraints validate that the mapping is done in a way that is intended by the semantic description of the profile's modeling elements.

8.1.1 Type mapping from SDL AS1 to MOF

The abstract syntax of SDL is defined by an abstract grammar [ITU06b]. For example, the abstract syntax for a *Channel-definition* defining a channel between two gates is:

Channel-definition :: *Channel-name*
 [**NODELAY**]
 Channel-path-set

This defines the domain for the composite object named *Channel-definition*. This object consists of further three sub-components. The definition

Channel-name = *Name*
Name :: *Token*

expresses that the *Channel-name* resolves into a *Token*. This composite object can also be perceived as a tree with some root node.

To be able to compare the values of two data types, both must be of the same data type. Therefore, all elementary types of the SDL AS1, which do not resolve into other types, are converted to an appropriate OCL data type. As noted, the abstract syntax of SDL can be regarded as a named composite object (a tree) defining a set of sub components. An object might also be of some elementary (non-composite) domains. In the context of SDL, these are: *Token*, *Nat*, *Quotation*, and *Set*. These elementary objects are mapped to an OCL type.

8.1.1.1 NATURAL OBJECTS

Example:

Number-of-instances :: *Nat [Nat]*

Number-of-instances denotes a composite domain containing one mandatory natural (Nat) value and one optional natural ([Nat]) denoting respectively the initial number and the optional maximum number of instances. A Nat is mapped to a non-negative Integer value.

convert: Nat -> Integer
context convert(nat: Nat) : Integer
post: result >=0

8.1.1.2 QUOTATION OBJECTS

These are represented as any bold face sequence of uppercase letters and digits.

Example:

Channel-definition :: *Channel-name*
 [**NODELAY**]
 Channel-path-set

A channel may be delaying or not. This is denoted by an optional quotation **NODELAY**. This quotation is mapped to a String that is associated with the respective compound object. If no composite object is defined in the AS, the name of the String is constructed with the first name of its containing composite object name with suffix *-kind*. For example, the name of the String containing the Quotation object **NODELAY** for a *Channel-definition* would be *Channel-kind*.

convert: Quotation -> String

8.1.1.3 UNSPECIFIED OBJECTS AND REPETITIONS

The abstract syntax appends the postfix operator *-set* denoting a set (un-ordered collection of distinct objects).

Example:

Agent-graph :: *Agent-start-node State-node-set*

An Agent-graph consists of an Agent-start-node and a set of State-nodes. A *-set* of the abstract syntax is mapped to a Set. A Set is an un-ordered collection of distinct object types.

convert: -set -> Set

The same mapping also applies to repetitions of composite object.

8.1.1.4 TOKEN OBJECTS

Token denotes the domain of tokens. This domain can be considered to consist of a potentially infinite set of distinct atomic objects for which no representation is required.

Example:

Name :: *Token*

A name consists of an atomic object so that any Name can be distinguished from any other name. A Token is mapped to a String.

convert: Token -> String

Each Token being an SDL name shall follow the EBNF grammar given in Z.100. An *Identifier* that addresses each distinct object within an SDL specification is also mapped to a String which is compatible to the UML notation for qualified names.

Identifier :: *Qualifier Name*
Qualifier = *Path-item +*
Path-item = *Package-qualifier | ...*
Package-qualifier :: *Package-name*
Package-name = *Name*
Name :: *Token*

convert: Qualifier, Name -> String

context convert(Qualifier: q[1..], Name: n): String*

post: let s:String="" in result = q->forAll(s->concat(convert(q))->concat("::"))->concat(convert(n))

8.1.1.5 UML CONSTRAINTS

UML Constraints are specified by means of a ValueSpecification. The ValueSpecification of a constraint is mapped to a String.

convert: Constraint -> String

context convert(c: Constraint): String

post: result = c.specification.stringValue()

8.1.2 OCL Constraints for AS1

The Figure 82 shows the conceptual outline of the type mapping and constrained tree. Dotted lines imply a type conversion. Underlined expressions denote OCL constraints applied to the object tree. The given constraints are only exemplary.

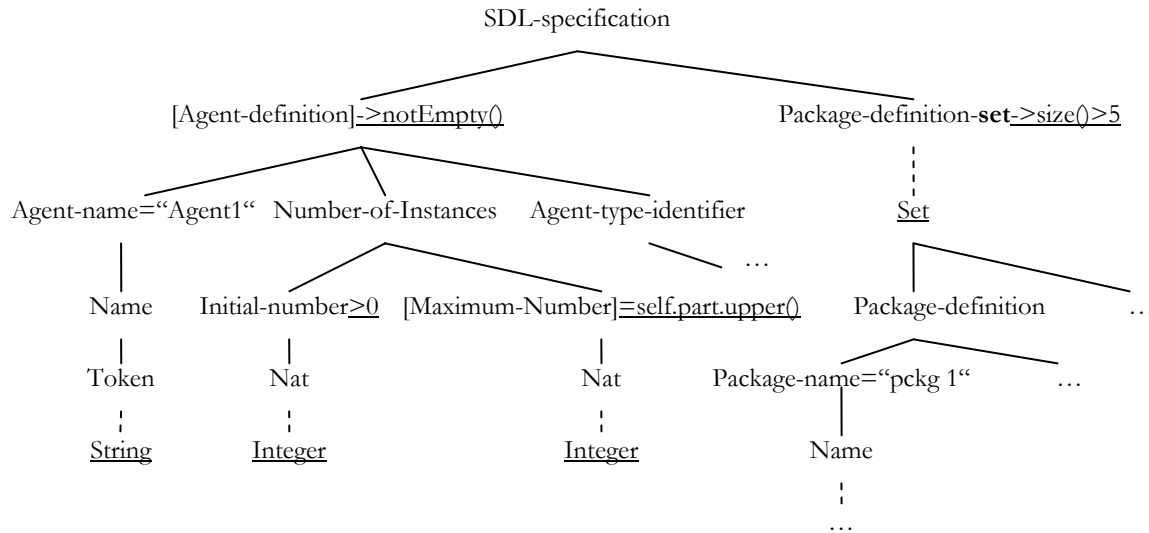


Figure 82: OCL Constraints on AS1 Composite Object Tree

During translation from the concrete SDL syntax to an executable system, there are several transformation steps applied (the concrete steps are not described here; it is referred to the appendices of Z.100 [ITU02a]). The abstract syntax AS1 is a composite object tree and defines an *SDL-specification* as root. From this, the tree is traversed by means of the defined objects within the system description. It is assumed that each object within this tree can be constrained by means of OCL. For this, type conformance has to be assured first. As described in Section 8.1.1, some type mappings are defined for a mapping to OCL compliant types. For example, a *Nat* within the tree is mapped to a non-negative *Integer*. However, a concrete mapping is not provided; instead, it is assumed that such a mapping is already available, so that all constraint qualifiers are valid (however, if the type compliance would fail, none of the constraint qualifiers is correct and the mapping is considered invalid).

The type mapping must not only be provided on leaf objects. Objects defining a set are implicitly mapped to an OCL set that is shown at the *Package-definition-set* object. The Bag itself contains the set of *Package-definitions* which itself decomposes into several objects. If the resolution path down to the tree leaf is unambiguous, it is assumed that derived attributes are available. For instance, although the *Agent-name* is resolved into a *Token* it is assumed that this is also done implicitly at parent objects. Therefore, the correct navigation through such a composite tree would be *Agent-name.Name.Token = "Agent1"*. However, the shorthand notation *Agent-name = "Agent1"* is allowed.

8.1.3 Example - Mapping the SDL AS1 for Channel Definition to MOF

Nevertheless, OCL constraints require some navigation on the composite objects. This is briefly illustrated in the following. For example, the abstract grammar of a channel definition in Z.100 main body is

Abstract grammar

Channel-definition :: *Channel-name*
 [NODELAY]
Channel-path-set

Channel-path :: *Originating-gate*
Destination-gate
Signal-identifier-set

Originating-gate = *Gate-identifier*

Destination-gate = *Gate-identifier*

Gate-identifier = *Identifier*

Agent-identifier = *Identifier*

Channel-name = *Name*

The mapping should map the following excerpt of the composite object tree of the AS1 to a metamodel that is type-conformant and navigable. The result is shown in Figure 83;

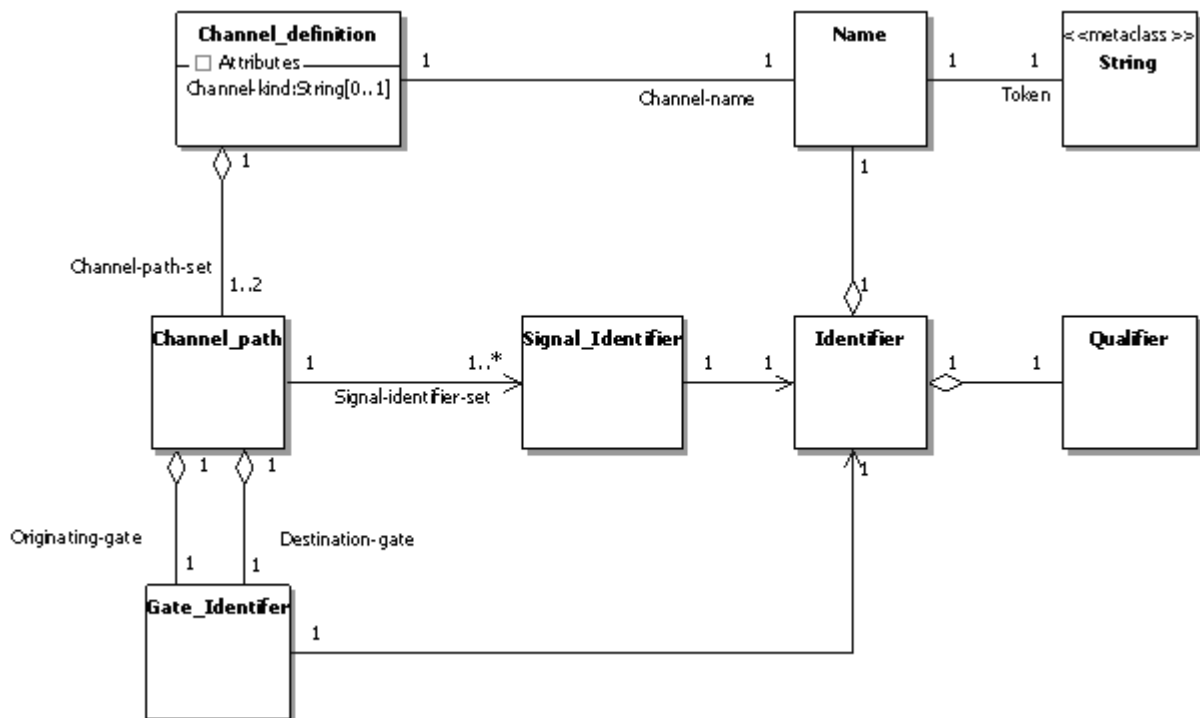


Figure 83: Abstract Grammar mapped to Metamodel

Note, this metamodel defines only an excerpt, as the associations of the metaclass Qualifier are omitted. In this example, *-set* are mapped to multiplicities 0..*. Optional components are mapped to multiplicities 0..1. However, quotation objects like *NODELAY* are currently mapped to a String with 0..1 multiplicity with the first name of its containing composite object and a *-kind* suffix. If the quotation does not apply (e.g. a channel that does not have the quotation *NODELAY*), the String property is empty.

Additionally, it is presumed that derived attributes are available in each class object, e.g. the (partially given) abstract syntax for the *Channel-definition* is

```

Channel-definition      ::  Channel-name
Channel-name           =   Name
Name                   ::  Token

```

To validate that the *Channel-name* is not empty, the correct navigation and constraint is *Channel-definition::Channel-name.Name.Token->notEmpty()*. The *Channel-name* resolves into a *Token* that is mapped to a String. Hence, as the composite object tree resolution path is unique and unambiguous, the expression *Channel-definition::Channel-name->notEmpty()* is also considered as valid.

8.2 OCL-based Mapping to SDL-2000

The mapping is described for the UML CS model elements which do not have a concrete textual representation. That is, when a UML CS stereotyped modeling element is to be mapped, a set of qualifiers is specified to assert that values are assigned to the corresponding SDL abstract syntax of an element. Therefore, the semantics and definitions of the various modeling elements are derived from the SDL construct. AS1 composite objects not being constrained by an OCL expression may have an optional assigned value; for instance, the start node of a state machine may have a name assigned or not. For clarity, the OCL constraints, UML metaclasses and stereotyped attributes are underlined. UML CS model elements with a concrete textual notation are not included in this mapping as the syntax is derived from SDL [ITU02b].

An operation with the navigation such as *self.part.upper()* shown in Figure 82, is only valid when a context is specified. In the following, the context is denoted by the stereotype name labeled as the heading. Thus, the navigation is always originating from the specified stereotype. The currently valid SDL identifier is retrieved from the stereotype's *qualifiedname()* operation.

8.2.1 Architecture

As the model repository of the UML tool is non-navigable, the following is expressed informally. The SDL-specification defines an optional *Agent-definition* or a *Package-definition-set*. The *Agent-definition* may be a system.

```

SDL-specification      ::  [Agent-definition] = "Agent definition with type
                               of kind system in model repository"
                               Package-definition-set = "All packages in the model repository"

```

8.2.1.1 PACKAGE

A package is mapped to a *Package-definition*. Its attributes are mapped according the following constraints:

```

Package-definition    ::  Package-name = package.name
                          Package-definition-set = package.nestedPackage
                          Data-type-definition-set =
                              package.packagedElement->select(e | isStereotypedBy(e, dataType))
                          Syntype-definition-set = package.packagedElement->
                              select(e | isStereotypedBy(e, constant))
                          Signal-definition-set =
                              package.packagedElement->select(e | isStereotypedBy(e, signal))

```

```

Exception-definition-set -> isEmpty()
Agent-type-definition-set = package.base Package.packagedElement->
  select(a | isStereotypedBy(a,block))->
  union(package.base Package.packagedElement ->
  select(a | isStereotypedBy(a,process))->
  union(package.base Package.packagedElement ->
  select(a | isStereotypedBy(a,system))
Composite-state-type-definition-set =
  package.packagedElement ->select(s|
  isStereotypedBy(s, state))->select(s | s.isComposite)
Procedure-definition-set =
  package.packagedElement -> select(s|isStereotypedBy(s, operation))

```

8.2.1.2 SYSTEM

A system is mapped to an Agent-type-definition. This assures that there shall be the correct quotation *SYSTEM*. The optional Agent-type-identifier is used to implement inheritance. The superClass is a Bag, however, as multiple inheritance is not allowed, the first superClass specifies the parent of this Agent-type. Formal parameters for agents are not supported. The Data-type-definition-set is defined by all objects and value types that are nested inside of this agent. The Syntype-definition-set is defined by all constant data types; the Signal-definition-set is defined by the nestedClassifier's stereotypes of type signal. The Timer-definition-set is defined by the nestedClassifiers with stereotype timer. Exception are not supported, therefore, the composite object Exception-definition-set is empty.

```

Agent-type-definition::
  Agent-type-name = system base Class.name
  Agent-kind = "SYSTEM"
  [ Agent-type-identifier ] = system.base Class.superClass[0].qualifiedName
  Agent-formal-parameter*-> isEmpty()
  Data-type-definition-set = system.base Class.nestedClassifiers->
    select(d | d.isStereotypedBy(d,value))->
    union(system.base Class.nestedClassifiers ->
    select(d | d.isStereotypedBy(d,object))
  Syntype-definition-set = system.base Class.ownedAttribute->
    select(d | d.isStereotypedBy(d,constant))
  Signal-definition-set = system.base Class.nestedClassifier->
    select(s | isStereotypedBy(s,signal))
  Timer-definition-set = system.base Class.nestedClassifier->
    select(t |isStereotypedBy(t,timer))
  Exception-definition-set -> isEmpty()
  Variable-definition-set = system.base Class.ownedAttribute->
    select(d | d.isStereotypedBy(d,value))->
    union(system.base Class.ownedAttribute->
    select(d | d.isStereotypedBy(d,object))
  Agent-type-definition-set = system.base Class.nestedClassifier->
    select(a | isStereotypedBy(a,block))->
    union(system.base Class.nestedClassifier->
    select(a | isStereotypedBy(a,process))
  Composite-state-type-definition-set -> isEmpty()
  Procedure-definition-set = system.base Class.ownedOperation
  Agent-definition-set = system.base Class.part
  Gate-definition-set = system.base Class.ownedPort
  Channel-definition-set = system.base Class.ownedConnector
  [ State-machine-definition ] -> isEmpty()

```

8.2.1.3 BLOCK

The block extended Class is mapped similar to a system. The difference is the Agent-kind quotation that specifies this agent.

Agent-type-definition::

```

Agent-type-name = block.base Class.name
Agent-kind = "BLOCK"
[ Agent-type-identifier ] = block.base Class.superClass[0].qualifiedName
Agent-formal-parameter* -> isEmpty()
Data-type-definition-set = block.base Class.nestedClassifiers->
select(d | d.isStereotypedBy(d,value))->
union(block.base Class.nestedClassifiers ->
select(d | d.isStereotypedBy(d,object))
Syntype-definition-set = block.base Class.ownedAttribute->
select(d | d.isStereotypedBy(d,constant))
Signal-definition-set = block.base Class.nestedClassifier->
select(s | isStereotypedBy(s,signal))
Timer-definition-set = block.base Class.nestedClassifier->
select(t | isStereotypedBy(t,timer))
Exception-definition-set -> isEmpty()
Variable-definition-set = block.base Class.ownedAttribute->
select(d | d.isStereotypedBy(d,value))->
union(block.base Class.ownedAttribute->
select(d | d.isStereotypedBy(d,object))
Agent-type-definition-set = block.base Class.nestedClassifier->
select(a | isStereotypedBy(a,block))->
union(block.base Class.nestedClassifier->
select(a | isStereotypedBy(a,process))
Composite-state-type-definition-set -> isEmpty()
Procedure-definition-set = block.base Class.ownedOperation
Agent-definition-set = block.base Class.part
Gate-definition-set = block.base Class.ownedPort
Channel-definition-set = block.base Class.ownedConnector
[ State-machine-definition ] -> isEmpty()

```

8.2.1.4 PROCESS

The process extended Class is mapped similar to a system. The difference is the Agent-kind quotation that specifies this agent. In addition, a process defines a state machine.

Agent-type-definition::

```

Agent-type-name = process base Class.name
Agent-kind = "PROCESS"
[ Agent-type-identifier ] = process.base Class.superClass[0].qualifiedName
Agent-formal-parameter* -> isEmpty()
Data-type-definition-set = process.base Class.nestedClassifiers->
select(d | d.isStereotypedBy(d,value))->
union(process.base Class.nestedClassifiers ->
select(d | d.isStereotypedBy(d,object))
Syntype-definition-set = process.base Class.ownedAttribute->
select(d | d.isStereotypedBy(d,constant))
Signal-definition-set = process.base Class.nestedClassifier->
select(s | isStereotypedBy(s,signal))
Timer-definition-set = process.base Class.nestedClassifier->
select(t | isStereotypedBy(t,timer))
Exception-definition-set -> isEmpty()
Variable-definition-set = process.base Class.ownedAttribute->
select(d | d.isStereotypedBy(d,value))->
union(process.base Class.ownedAttribute->
select(d | d.isStereotypedBy(d,object))
Agent-type-definier-set = process.base Class.nestedClassifier->

```

```

select(a | isStereotypedBy(a,block)->
union(process.base Class.nestedClassifier->
select(a | isStereotypedBy(a,process))
Composite-state-type-definition-set =
  process.base Class.ownedBehavior->
  excluding(process.base Class.ownedOperation->collect(method))
Procedure-definition-set =
  process.base Class.ownedBehavior->
  intersection(process.base Class.ownedOperation->collect(method))
Agent-definition-set = process.base Class.part
Gate-definition-set = process.base.Class.ownedPort
Channel-definition-set = process.base Class.ownedConnector
[ State-machine-definition ]
State-machine-definition :: State-name = process.base Class.classifierBehavior.name
                          Composite-state-type-identifier = process.base Class.classifierBehavior

```

8.2.1.5 STRUCTUREDCLASSIFIER

For a StructuredClassifier (or for the *instance* stereotype, alternatively), there shall be an Agent-definition which matches its name with the part property name (note that all agent types are specializations of a StructuredClassifier). If the names are matching, the type and the number of instances must also match:

```

StructuredClassifier.part->forall (part: Property |
Agent-definition :: Agent-name = part.name
                  Number-of-instances
                  Agent-type-identifier = part.type
Number-of-instances :: Initial-number [Maximum-number]
Initial-number = Nat = part.lower
Maximum-number = Nat = part.upper
                  )

```

8.2.1.6 OPERATION

An operation stereotype is mapped to a procedure-definition if it is contained in an active class (like an agent). It is mapped to an Operation-Signature if it is contained in a non-active class, see 8.2.4.2.

if operation.base Operation.class.isActive then

```

Procedure-definition :: Procedure-name = operation.base Operation.name
                    Procedure-formal-parameter* =
                      operation.base Operation.ownedParameter
                    [Result] = operation.base Operation.type.qualifiedName
                    [Procedure-identifier] =
                      operation.base Operation.general[0].qualifiedName
                    Data-type-definition-set =
                      operation.base Operation.method.ownedAttribute
                    Syntype-definition-set =
                      operation.base Operation.method.ownedAttribute->
                      select(d.ocIsKindOf(constant))
                    Variable-definition-set =
                      operation.base Operation.method.ownedAttribute->
                      reject(d.ocIsKindOf(constant))
                    Composite-state-type-definition-set -> isEmpty()
                    Procedure-definition-set -> isEmpty()
                    Procedure-graph = operation.base Operation.method
Procedure-name = Name
Procedure-formal-parameter = In-parameter
                          | Inout-parameter

```

```

|      Out-parameter
Result      ::      Sort-reference-identifier
endif

```

8.2.1.7 PARAMETER

Depending on the direction of the parameters a parameter is either mapped to an In-, Inout- or –Out parameter.

```

In-parameter      ::
if parameter.base Parameter.direction = ParameterDirectionKind::in then
    Parameter
endif
Inout-parameter   ::
if parameter.base Parameter.direction = ParameterDirectionKind::inout then
    Parameter
endif
Out-parameter     ::
if parameter.base Parameter.direction = ParameterDirectionKind::out then
    Parameter
endif
Parameter         ::      Variable-name = parameter.base Parameter.name
Sort-reference-identifier = parameter.base Parameter.type

```

8.2.1.8 INTERFACE

An interface is mapped to an Interface-definition. Multiple inheritance is not supported.

```

Interface-definition      ::      Sort = interface.base Interface.name
Data-type-identifier* = interface.base Interface.general[0]
Data-type-definition-set =
    interface.base Interface.ownedAttributes ->
    reject(d.oclIsKindOf(constant))
Syntype-definition-set =
    interface.base Interface.ownedAttributes->
    select(d.oclIsKindOf(constant))
Exception-definition-set -> isEmpty()

```

8.2.1.9 PORT

A non-dynamic port is mapped to a Gate-definition. The mapping for a dynamic port is described informally due to the complex mapping to an equivalent SDL gate.

```

if not port.isDynamic then
Gate-definition      ::      Gate-name = port.base Port.name
In-signal-identifier-set =
    port.base Port.providedInterface.ownedReception->
    collect(signal)
Out-signal-identifier-set =
    port.base Port.requiredInterface.ownedReception->
    collect(signal)
Gate-name            =      Name
In-signal-identifier =      Signal-identifier
Out-signal-identifier =      Signal-identifier
endif

```

Mapping of a dynamic port group

SDL has no corresponding language construct like dynamic ports. However, a semantically equivalent system description can be generated. The main idea is to generate multiple agent type definitions based on the amount of dynamic ports connected to a group. That is, the mapping should first enumerate the amount of different dynamic ports used within an agent definition. Based on this amount, several agent types with a different amount of SDL ports are created. They only differ in their amount of ports. For each dynamic port a unique name is assigned. For instance, the dynamic ports $nc(0)$ is assigned $port_1$, port $nc(1)$ is assigned $port_2$ and so on. If the property *instances* of a dynamic port is read, this property should be replaced by a constant value specifying the amount of available dynamic ports. If a dynamic port is addressed by means of a variable index, decisions have to be added and the input or output action has to be replicated.

The downside of this approach is that when agents use various dynamic port groups, the amount of agent types, which have to be generated, can become very huge.

8.2.1.10 CHANNEL

A channel is mapped to a Channel-definition. If the channel is defined as a non-delaying channel, the appropriate quotation NODELAY is present. Note that this introduces a new composite object named *Channel-kind* which is not part of the SDL AS1 in Z.100. However, this shall not impose any semantic difference. Note that the Channel-name does not have any constraint. This means, that the mapping for this object is unspecified. So there may be a name assigned or not. As channel names cannot be used for addressing signals, channel names are only for clarity to the user.

<i>Channel-definition</i>	::	<i>Channel-name</i> <u><i>channel.delay implies Channel-kind = "NODELAY"</i></u>
<i>Channel-path</i>	::	<u><i>if channel.signalList0->size()>0 then</i></u> <i>Originating-gate = channel.end[0].role</i> <i>Destination-gate = channel.end[1].role</i> <i>Signal-identifier-set = channel.signalList0</i> <u><i>endif</i></u>
<i>Channel-path</i>	::	<u><i>if channel.signalList1->size()>0 then</i></u> <i>Originating-gate = channel.end[1].role</i> <i>Destination-gate = channel.end[0].role</i> <i>Signal-identifier-set = channel.signalList1</i> <u><i>else</i></u> <i>Originating-gate -> isEmpty()</i> <i>Destination-gate -> isEmpty()</i> <i>Signal-identifier-set -> isEmpty()</i> <u><i>endif</i></u>
<i>Originating-gate</i>	=	<i>Gate-identifier</i>
<i>Destination-gate</i>	=	<i>Gate-identifier</i>
<i>Gate-identifier</i>	=	<i>Identifier</i>
<i>Agent-identifier</i>	=	<i>Identifier</i>
<i>Channel-name</i>	=	<i>Name</i>

8.2.1.11 SIGNAL

A signal is mapped to a Signal-definition. If there are any attributes defined for this signals, the corresponding types are mapped to SDL sorts.

<i>Signal-definition</i>	::	<i>Signal-name = signal.base Signal.name</i> <i>Sort-reference-identifier* =</i> <u><i>signal.base Signal.ownedAttribute->collect(type)</i></u>
<i>Signal-identifier</i>	=	<i>Identifier</i>

Signal-name = *Name*

8.2.1.12 SIGNALLIST

A signalList contains the explicit definition of multiple signals as shorthand notation. Each contained signal of the signalList is mapped to a distinct signal.

signalList.ownedSignal->forall(s: Signal |
Signal-definition = s
)

8.2.2 Behavior

8.2.2.1 STATEMACHINE

A stateMachine is mapped to a Procedure-graph if the stateMachine is the implementation of an operation.

if stateMachine.base StateMachine..specification->oclIsTypeOf(Operation) then
Procedure-graph :: [On-exception] -> isEmpty()
 [Procedure-start-node]
State-node-set = stateMachine.base StateMachine.region[1].
subvertex->select(v | isStereotypedBy(v, state))
Free-action-set = stateMachine.base StateMachine.region[1].
subvertex->select(v | isStereotypedBy(v, state))
->collect(transition)
Exception-handler-node-set -> isEmpty()
Procedure-start-node :: [On-exception]
 Transition =
self.region[0].subvertex.select(v |
v.oclIsKindOf(Pseudostate))->
select(p: Pseudostate | p.kind=PseudostateKind::initial)
Procedure-identifier = *Identifier* = self.base StateMachine.general

else if stateMachine.base StateMachine.specification->isEmpty()

Composite-state-type-definition :: *State-type-name* = stateMachine.base StateMachine.name
 [*Composite-state-type-identifier*] =
stateMachine.base StateMachine.general
*Composite-state-formal-parameter** = stateMachine.
base StateMachine.ownedParameter
State-entry-point-definition-set -> isEmpty()
State-exit-point-definition-set -> isEmpty()
Gate-definition-set -> isEmpty()
Data-type-definition-set = stateMachine.base StateMachine.
nestedClassifier->select(d | d.oclIsKindOf(Datatype))
Syntype-definition-set = stateMachine.base StateMachine.
nestedClassifier->select(d | d.oclIsKindOf(Datatype))
->select(c | isStereotyped(c, constant))
Exception-definition-set -> isEmpty()
Composite-state-type-definition-set
stateMachine.base StateMachine.ownedOperation->
select(d | not d->exist(stateMachine.
base StateMachine.context.ownedBehavior))
Variable-definition-set =
stateMachine.base StateMachine.ownedAttribute
Procedure-definition-set =

stateMachine.base StateMachine.ownedOperation->
select(d | d->exist(stateMachine.
base StateMachine.context.ownedBehavior))
 [Composite-state-graph | State-aggregation-node]

endif

8.2.2.2 REGION

A region is not mapped to SDL and is therefore ignored.

8.2.2.3 START

A start extended pseudostate is mapped to a State-start-node. Exceptions are not supported. The Transition is defined by the outgoing transition of the pseudostate.

State-start-node:: [On-exception] -> isEmpty()
 [State-entry-point-name] -> start.base Pseudostate.name
 Transition = start.base Pseudostate.outgoing

8.2.2.4 STATE

A state stereotype is mapped to a State-node. The deferrableTrigger defines the Save-signalset. Spontaneous- and the Continuous-signal nodes are calculated from the transition guards and triggers. An asterisk state and a state list is transformed according to the steps listed in Z.100 [ITU02a] Annex F2 Chapter 3.

State-node:: State-name = state.name
 [On-exception] -> isEmpty()
 Save-signalset = state.deferrableTrigger
 Input-node-set = state.outgoing->select(t.trigger->notEmpty())
 Spontaneous-transition-set =
 state.outgoing->select(t.trigger->isEmpty() and t.guard->isEmpty())
 Continuous-signal-set =
 state.outgoing->select(t.trigger->isEmpty() and t.guard->notEmpty())
 Connect-node-set ->isEmpty()
 [Composite-state-type-identifier] ->isEmpty()

8.2.2.5 COMPOSITESTATE

A compositeState is mapped similar to a state. The difference is the Composite-state-type-identifier.

State-node:: State-name = state.name
 [On-exception] -> isEmpty()
 Save-signalset = state.deferrableTrigger
 Input-node-set = state.outgoing->select(t.trigger->notEmpty())
 Spontaneous-transition-set =
 state.outgoing->select(t.trigger->isEmpty() and t.guard->isEmpty())
 Continuous-signal-set =
 state.outgoing->select(t.trigger->isEmpty() and t.guard->notEmpty())
 Connect-node-set = state.connectionPoint->
 select(c | c.kind = PseudostateKind::exitPoint)
 [Composite-state-type-identifier] = state.submachine

8.2.2.6 ENTRYPOINT

An endpoint extended pseudostate is mapped to a State-start-node. Exceptions are not supported; the Transition is defined by the outgoing transition of the endpoint. There is no semantic difference to a start extended pseudostate.

State-start-node :: [On-exception] -> isEmpty()
[State-entry-point-name] -> endpoint.base Pseudostate.name
 Transition = start.base Pseudostate.outgoing

8.2.2.7 EXITPOINT

An endpoint pseudostate is mapped to a Named-return-node.

Named-return-node :: State-exit-point-name = endpoint.base Pseudostate.name

8.2.2.8 TRANSITION

The UML CS profile introduces a more powerful receive signal action as it is possible with SDL. The main difference lies in the *FROM* and *VIA* clause, as there is currently no equivalent instruction available in SDL. The <input> node combines the SDL state machine elements Continuous Signal, Guard and a modified Input node. If no *FROM* or *VIA* clause is given the following rules apply for the mapping to SDL:

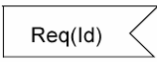
- If the trigger is empty in an Receive Signal node, the corresponding node in SDL is the Continuous Signal

SDL/PR NOTATION	SDL/GR NOTATION
PROVIDED <self.guard>;	

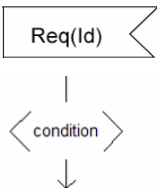
Continuous-signal :: [On-exception] -> isEmpty()
Continuous-expression
[Priority-name]
 Transition

Continuous-expression = transition.guard.specification
if transition.prioritized then
Priority-name = 1
else
Priority-name = 0
endif

- If the trigger is not empty and the guard is empty, the corresponding node in SDL is the Input node.

SDL/PR NOTATION	SDL/GR NOTATION
INPUT <self.trigger>;	

- If both trigger and guard are not empty, the corresponding node in SDL is the Input node with a subsequent Enabling Condition.

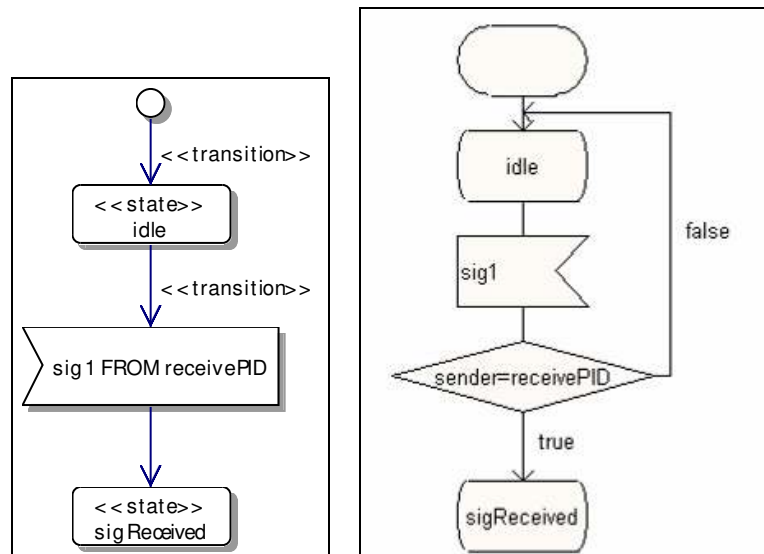
SDL/PR NOTATION	SDL/GR NOTATION
INPUT <self.trigger>; PROVIDED <self.guard>;	

Input-node :: transition.priorized implies *Input-kind* = "PRIORITY"
Signal-identifier = transition.trigger.signal
[Variable-identifier]*
[Provided-expression]
[On-exception] -> isEmpty()
Transition = transition.base *Transition.effect*

Variable-identifier = transition.base *Transition.trigger.signal.ownedAttribute*

Provided-expression = transition.base *Transition.guard*

If the additional attributes for a signal input *from* and *via* are specified, the mapping is not possible while conserving the semantics. A SignalEvent *via* cannot be mapped to SDL, as there is no language support for determining the gate at which a signal has been received. Nevertheless, the *from* attribute can provisionally be mapped. Figure 84 shows a possible mapping:



(a) UML CS *Input From* (b) SDL *Input From* Mapping Approach

Figure 84: Provisional Approach mapping UML CS Input with FROM Clause to SDL

This mapping is not semantically equivalent. The SDL model shown on the right can only determine the signal's origin after its consumption using the input node. Then the implicit assigned *sender* variable is evaluated which contains the Process Identification of the process from which the signal has been received. This is different from the UML CS state machine model shown on the left. The FROM clause prevents the consumption of the signal *sig1* if it is not originated from a process which *Process Identification* is equal to value of variable *receivePID*. Unfortunately, it is not possible in SDL to re-insert a consumed signal to the input queue while conserving the correct *sender* variable.

The following OCL operation *collectAction* traverses an activity in the order specified by the control flow. Reaching a forking node or an end of control flow node terminates the collection. The operation returns an ordered set of collected Activities.

```

UMLCS::collectAction(flow: ControlFlow): OrderedSet
let nextNode: Activity = flow.target in
  if isStereotyped(nextNode, return) or
    isStereotyped(nextNode, choice)
    result=OrderedSet {}
  else
    result=nextNode->append(collectActions(nextNode.outgoing[1]))

```

```

Transition                ::   Graph-node* ≡ collectAction(transition.base Transition.effect)
                             ( Terminator | Decision-node )
Graph-node                ::   ( Task-node
                               | Output-node
                               | Create-request-node
                               | Call-node
                               | Compound-node
                               | Set-node
                               | Reset-node )
                             [On-exception] -> isEmpty()
Terminator                ::   (
if isStereotypedBy(transition.base Transition.target, state)
                             Nextstate-node
else if isStereotypedBy(transition.base Transition.target, stop)
                             | Stop-node
else if isStereotypedBy(transition.base Transition.target, return)
                             | Return-node
else if isStereotypedBy(transition.base Transition.target, merge)
                             | Join-node
else
                             | Continue-node
                             | Break-node
                             | Raise-node )
                             [On-exception] -> isEmpty()

```

8.2.2.9 STOP

A stop extended pseudostate is mapped to a Stop-node.

```

Stop-node                ::   ( )

```

8.2.2.10 DECISION

The outgoing transition guards from this decision pseudostate node shall have a common question (left hand side) and several possible answers (right hand side). This cannot be expressed in OCL.

```

Decision-node          ::  Decision-question = "The common LHS of all guards"
                        [On-exception] -> isEmpty()
                        Decision-answer-set
                        [Else-answer]
Decision-question      =  Expression
                        |  Informal-text
Decision.base PseudoState.outgoing ->forAll(t |
Decision-answer        ::  Range-condition = t->reject(t.guard="else").guard
                        Transition = t->reject(guard="else")
)
Else-answer            ::  Transition = decision.base PseudoState.outgoing->
                        select(guard="else")

```

8.2.2.11 MERGE

A merge pseudostate is mapped to a Free-action node with a named Connector. Each transition connecting to this pseudostate defines a Join-node with the same name as a Terminator.

```

Free-action            ::  Connector-name = merge.base Pseudostate.name
                        Transition
Connector-name          =  Name
Transition              ::  Graph-node* = collectAction(merge.
                        base Pseudostate.outgoing[0].effect)
                        ( Terminator | Decision-node )
Graph-node             ::  (
if isStereotypedBy(merge.base Pseudostate.outgoing[1], task)
                        Task-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], output)
                        | Output-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], createObject)
                        | Create-request-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], callOperation)
                        | Call-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], compoundNode)
                        | Compound-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], setTimer)
                        | Set-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], resetTimer)
                        | Reset-node )
endif
                        [On-exception] -> isEmpty()
Terminator              ::  (
if isStereotypedBy(merge.base Pseudostate.outgoing[1], state)
                        Nextstate-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], stop)
                        | Stop-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], return)
                        | Return-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], merge)
                        | Join-node
else if isStereotypedBy(merge base Pseudostate..outgoing[1], transition)
                        | Continue-node
else if isStereotypedBy(merge.base Pseudostate.outgoing[1], break)
                        | Break-node

```

```

else
| Raise-node ) -> isEmpty()
endif
[On-exception] -> isEmpty()

```

8.2.2.12 HISTORY

A history is mapped to a Nextstate-node with quotation HISTORY.

```

Nextstate-node      ::  State-name = history.base Pseudostate.name
                       [Nextstate-parameters]
Nextstate-parameters  ::  [Expression]* -> isEmpty()
                       [State-entry-point-name] -> isEmpty()
                       Nextstate-kind = "HISTORY"

```

8.2.2.13 SEQUENCENODE

A sequenceNode defines a Compound-node.

```

if sequenceNode.activity->isEmpty() then
Compound-node      ::  Connector-name = sequenceNode.base SequenceNode.name
                       Variable-definition-set =
                           sequenceNode.base SequenceNode.variable
                       [ Exception-handler-node ] -> isEmpty()
                       Init-graph-node* =
                           sequenceNode.base SequenceNode.executableNode[0]
                       Transition
                       Step-graph-node* =
                           sequenceNode.base SequenceNode.executableNode
Init-graph-node    =  Graph-node
Step-graph-node    =  Graph-node
endif

```

8.2.2.14 OUTPUT

An output extended SendSignalAction is mapped to an Output-node. The signal and the argument of this action are mapped. The optional destination of the signal including the outgoing port is also mapped.

```

Output-node      ::  Signal-identifier = output.base SendSignalAction.signal
                       [Expression]* = output.base SendSignalAction.argument
                       output.via->notEmpty() implies [Signal-destination]->isEmpty()
                       Direct-via
Signal-destination  =  output base SendSignalAction.target
Direct-via         =  Gate-identifier-set = output.via

```

8.2.2.15 OPERATIONCALL

An operationCall is mapped to a Call-node if the target procedure is local.

```

if not isRemote(operationCall, operationCall.base CallOperationAction.operation)
Call-node      ::  Procedure-identifier =
                       operationCall.base CallOperationAction.operation
                       [Expression]* =
                       operationCall.base CallOperationAction.argument
endif

```

8.2.2.16 CREATEOBJECT

The createObject is mapped to a Create-request-node if the class which is to be instantiated is an agent type, e.g. a process.

```

if createObject.classifier->oclIsKindOf(agent)
Create-request-node      ::    [Variable-identifier] =
                           createObject.base CreateObjectAction.result
                           Agent-identifier =
                           createObject.base CreateObjectAction.classifier
                           [Expression]* = createObject.base CreateObjectAction.attributes
endif

```

8.2.2.17 NOOPERATION

This model element is not mapped and is therefore ignored.

8.2.2.18 RETURN

A return is either mapped to an Action-return-node if it does not define any arguments. It is mapped to a Value-return-node otherwise.

```

if return.base FinalFlow.argument->isEmpty()
Action-return-node      ::    ()
else
Value-return-node      ::    Expression = return.base FinalFlow.argument
endif

```

8.2.2.19 METHODSTART

A methodStart is mapped to a State-start-node. Exceptions are not supported; the Transition is defined by the outgoing transition of the pseudostate.

```

State-start-node::      [On-exception] -> isEmpty()
                       [State-entry-point-name] -> methodStart.base Pseudostate.name
                       Transition = methodStart.base Pseudostate.outgoing

```

8.2.2.20 METHODRETURN

A methodReturn is mapped to an Action-return-node and does not return any value.

```

Return-node             =    Action-return-node
Action-return-node     ::    ()

```

8.2.2.21 IF

An if ConditionalNode is mapped to a Decision-node.

```

Decision-node          ::    Decision-question
                           [On-exception] -> isEmpty()
                           Decision-answer-set
                           [Else-answer]
Decision-question      =    Expression = if.base ConditionalNode.clause->
                           reject(c.test="Always results true").test
                           | Informal-text
if.base ConditionalNode.clause->forall(c]

```



```

Decision-answer      :: ( Range-condition = if.base ConditionalNode.clause->
                        reject(c.test="Always results true").test
                        | Informal-text )
                        Transition = c.body
)
Else-answer          :: Transition = if.base ConditionalNode.clause->
                        select(c.test="Always results true").body

```

8.2.2.22 FOR

The for stereotype is mapped to a compound node. This compound node is defined as a replacement for a for-loop. The conditional loop is substituted by a decision-node that executes the body of the for loop while the run-condition evaluates to true. After the execution is finished, the foraction is executed. Then it jumps to the beginning of the compound node for the next iteration.

```

let connectorname: String = random() in
Compound-node       :: Connector-name = connectorname
                        Variable-definition-set = for.base LoopNode.loopVariable
                        [Exception-handler-node] ->isEmpty()
                        Init-graph-node* = for.base LoopNode.setupPart
                        Transition1
                        Step-graph-node* = for.foraction
Init-graph-node     = Graph-node
Step-graph-node     = Graph-node
Continue-node       :: Connector-name = connectorname

Transition1        :: Graph-node* ->isEmpty()
                    Decision-node

Decision-node      :: Decision-question = for.base LoopNode.test
                    [On-exception] -> isEmpty()
                    Decision-answer-set
                    [Else-answer]

Decision-question  = Expression
Decision-answer    :: Range-condition = "true"
                    Transition2

Transition2        :: Graph-node* = for.base.LoopNode.body
                    Terminator1

Terminator1       :: Join-node1
                    [On-exception]->isEmpty()

Join-node1        :: Connector-name = connectorname

```

8.2.2.23 REPEAT

The repeat stereotype is mapped to a compound-node with an embedded (while) compound-node1. This latter compound node is defined as a replacement for a while-loop (see 8.2.2.24).

```

let connectorname1: String = random() in
Compound-node       :: Connector-name = connectorname1
                        Variable-definition-set -> isEmpty()
                        [Exception-handler-node] ->isEmpty()
                        Init-graph-node* ->isEmpty()
                        Transition3
                        Step-graph-node* ->isEmpty()
Init-graph-node     = Graph-node

```

<i>Step-graph-node</i>	=	<i>Graph-node</i>
<i>Break-node</i>	::	<i>Connector-name</i> = <u><i>connectorname1</i></u>
<i>Transition3</i>	::	<i>Graph-node1</i> * -> <u><i>size()</i></u> =1 <i>Terminator2</i>
<i>Graph-node1</i>	::	<i>Compound-node1</i> [<i>On-exception</i>] -> <u><i>isEmpty()</i></u>
<i>Terminator2</i>	::	<i>Break-node</i> [<i>On-exception</i>]
<u><i>let connectorname: String = random() in</i></u>		
<i>Compound-node1</i>	::	<i>Connector-name</i> = <u><i>connectorname</i></u> <i>Variable-definition-set</i> -> <u><i>isEmpty()</i></u> [<i>Exception-handler-node</i>] -> <u><i>isEmpty()</i></u> <i>Init-graph-node</i> * -> <u><i>isEmpty()</i></u> <i>Transition1</i> <i>Step-graph-node</i> * -> <u><i>isEmpty()</i></u>
<i>Init-graph-node</i>	=	<i>Graph-node</i>
<i>Step-graph-node</i>	=	<i>Graph-node</i>
<i>Continue-node</i>	::	<i>Connector-name</i> = <u><i>connectorname</i></u>
<i>Transition1</i>	::	<i>Graph-node</i> * -> <u><i>isEmpty()</i></u> <i>Decision-node</i>
<i>Decision-node</i>	::	<i>Decision-question</i> = <u><i>repeat.base LoopNode.test</i></u> [<i>On-exception</i>] -> <u><i>isEmpty()</i></u> <i>Decision-answer-set</i> [<i>Else-answer</i>]
<i>Decision-question</i>	=	<i>Expression</i> <i>Informal-text</i>
<i>Decision-answer</i>	::	<i>Range-condition</i> = <u>"true"</u> <i>Transition2</i>
<i>Transition2</i>	::	<i>Graph-node</i> * = <u><i>repeat.base.LoopNode.body</i></u> <i>Terminator1</i>
<i>Terminator1</i>	::	<i>Join-node1</i> [<i>On-exception</i>]-> <u><i>isEmpty()</i></u>
<i>Join-node1</i>	::	<i>Connector-name</i> = <u><i>connectorname</i></u>

8.2.2.24 WHILE

The while stereotype is mapped to a compound node. This compound node is defined as a replacement for a while-loop. The conditional loop is substituted by a decision-node that executes the body of the while loop while the run-condition evaluates to true. After the execution is finished it jumps to the beginning of the compound node for next iteration.

<u><i>let connectorname: String = random() in</i></u>		
<i>Compound-node</i>	::	<i>Connector-name</i> = <u><i>connectorname</i></u> <i>Variable-definition-set</i> -> <u><i>isEmpty()</i></u> [<i>Exception-handler-node</i>] -> <u><i>isEmpty()</i></u> <i>Init-graph-node</i> * -> <u><i>isEmpty()</i></u> <i>Transition1</i> <i>Step-graph-node</i> * -> <u><i>isEmpty()</i></u>
<i>Init-graph-node</i>	=	<i>Graph-node</i>
<i>Step-graph-node</i>	=	<i>Graph-node</i>
<i>Continue-node</i>	::	<i>Connector-name</i> = <u><i>connectorname</i></u>

<i>Transition1</i>	::	<i>Graph-node*</i> <u>->isEmpty()</u> <i>Decision-node</i>
<i>Decision-node</i>	::	<i>Decision-question</i> \equiv <i>while.base LoopNode.test</i> [<i>On-exception</i>] <u>-> isEmpty()</u> <i>Decision-answer-set</i> [<i>Else-answer</i>]
<i>Decision-question</i>	=	<i>Expression</i>
		<i>Informal-text</i>
<i>Decision-answer</i>	::	<i>Range-condition</i> \equiv <u>"true"</u> <i>Transition2</i>
<i>Transition2</i>	::	<i>Graph-node*</i> \equiv <i>while.base LoopNode.body</i> <i>Terminator1</i>
<i>Terminator1</i>	::	<i>Join-node1</i> [<i>On-exception</i>]-> <u>isEmpty()</u>
<i>Join-node1</i>	::	<i>Connector-name</i> \equiv <u>connectorname</u>

8.2.2.25 CONTINUE

The continue node is mapped to the Continue-node.

<i>Continue-node</i>	::	<i>Connector-name</i> \equiv <u>continue.base OpaqueAction.name</u>
----------------------	----	---

8.2.2.26 BREAK, BREAKLABEL

The break and breaklabel nodes are mapped to a Break-node.

<i>Break-node</i>	::	<i>Connector-name</i> \equiv <u>break.base OpaqueAction.name</u>
-------------------	----	--

8.2.2.27 WRITESTRUCTURALFEATUREVALUEACTION

The WriteStructuralFeatureValueAction assigns a new value to a structural feature of an agent (a property). As this depends on the assigned value type this can be mapped to either an Assignment or an Assignment-attempt. The determination of the suitable mapping decision is up to the mapper.

<i>Task-node</i>	=	<i>Assignment</i> <i>Assignment-attempt</i>
<i>Assignment</i>	::	<i>Variable-identifier</i> \equiv <u>writeStructuralFeatureValueAction.</u> <u>base WriteStructuralFeatureValueAction.variable</u> <i>Expression</i> \equiv <u>writeStructuralFeautureValueAction.</u> <u>base WriteStructuralFeatureValueAction.variable</u>
<i>Assignment-attempt</i>	::	<i>Variable-identifier</i> \equiv <u>writeStructuralFeautureValueAction.</u> <u>base WriteStructuralFeatureValueAction.variable</u> <i>Expression</i> \equiv <u>writeStructuralFeatureValueAction.</u> <u>base WriteStructuralFeatureValueAction.variable</u>

8.2.2.28 WRITEVARIABLEVALUEACTION

The WriteVariableValueAction assigns a new value to a variable (a local variable). As this depends on the assigned value type this can be mapped to either an Assignment or an Assignment-attempt. The determination of the suitable mapping decision is up to the mapper.

<i>Task-node</i>	=	<i>Assignment</i> <i>Assignment-attempt</i>
<i>Assignment</i>	::	<i>Variable-identifier</i> ≡ <u><i>writeVariableValueAction.</i></u> <u><i>base WriteVariableValueAction.variable</i></u> <i>Expression</i> ≡ <u><i>writeVariableValueAction.</i></u> <u><i>base WriteVariableValueAction.variable</i></u>
<i>Assignment-attempt</i>	::	<i>Variable-identifier</i> = <u><i>writeVariableValueAction.</i></u> <u><i>base WriteVariableValueAction.variable</i></u> <i>Expression</i> ≡ <u><i>writeVariableValueAction.</i></u> <u><i>base WriteVariableValueAction.variable</i></u>

8.2.3 Timer

The following section describes the timer related activities. As SDL provides the same timer mechanism and activities as used in this profile simple mappings are possible.

8.2.3.1 TIMER

A timer extended metaclass is mapped to a Timer-definition. Its attributes define the optional Sort-reference-identifiers. Multiple timer expressions are transformed according to rule listed in Z.100 [ITU02a] Annex F2 Chapter 3.

<i>Timer-definition</i>	::	<i>Timer-name</i> ≡ <u><i>timer.base Signal.name</i></u> <i>Sort-reference-identifier*</i> ≡ <u><i>timer.base Signal.ownedAttribute</i></u>
<i>Timer-name</i>	=	<i>Name</i>

8.2.3.2 SETTIMER

<i>Set-node</i>	::	<i>Time-expression</i> ≡ <u><i>setTimer.timeout</i></u> <i>Timer-identifier</i> ≡ <u><i>setTimer.timer</i></u> <i>Expression*</i> ≡ <u><i>setTimer.timer.attribute</i></u>
<i>Timer-identifier</i>	=	<i>Identifier</i>
<i>Time-expression</i>	=	<i>Expression</i>

8.2.3.3 ACTIVE

<i>Timer-active-expression</i>	::	<i>Timer-identifier</i> ≡ <u><i>active.timer</i></u> <i>Expression*</i> ≡ <u><i>active.timer.attribute</i></u>
--------------------------------	----	---

8.2.3.4 RESETTIMER

<i>Reset-node</i>	::	<i>Timer-identifier</i> ≡ <u><i>resetTimer.timer</i></u> <i>Expression*</i> ≡ <u><i>resetTimer.timer.attribute</i></u>
-------------------	----	---

8.2.4 Data

The following section describes the mapping of data types. A property denotes a *slot* of a Class that specifies its attributes. A property can be marked as read-only: then it is a constant. Otherwise, it is a variable definition.

8.2.4.1 PROPERTY

If a property can be written, then it is mapped to a Variable-definition.

if not property.base Property.isReadOnly then

```
Variable-definition      ::  Variable-name = Property.name
                          Sort-reference-identifier = Property.dataType.qualifiedName
                          [ Constant-expression ] = Property.defaultValue
Variable-name           =  Name
endif
```

8.2.4.2 OPERATION

An operation that is not defined within an active class is mapped to a Dynamic- or Static-operation-signature. Operations of an active class are mapped in Section 8.2.1.6.

if not operation.base Operation.class.isActive then

```
Dynamic-operation-signature =  Operation-signature
Static-operation-signature  =  Operation-signature
Operation-signature        ::  Operation-name = operation.base Operation.name
                              Formal-argument* =
                              operation.base Operation.ownedParameter->
                              select(p | p.direction=ParameterDirectionKind::in)
                              [Result] = operation.base Operation.returnResult()
                              Identifier
Operation-name              =  Name
Formal-argument            =  Virtual-argument → isEmpty()
                              | Nonvirtual-argument
Virtual-argument           ::  Argument
Nonvirtual-argument        ::  Argument
Argument                   =  Sort-reference-identifier
endif
```

8.2.4.3 PRIMITIVE TYPE

A primitiveType is mapped to a Literal-Signature. The Name defines the Literal-name.

```
Literal-signature      ::  Literal-name = primitiveType.base PrimitiveType.name
                          Result
Literal-name           =  Name
```

8.2.4.4 VALUE

A value is mapped to a Value-data-type-definition. The operations define the operation-sets. The data type defines the Data-type-definition-set.

```
Value-data-type-definition  ::  Sort = value.base DataType.name
                              Data-type-identifier =
                              value.base DataType.general[0].qualifiedName
                              Literal-signature-set -> isEmpty()
```

Static-operation-signature-set \equiv
value.base DataType.ownedOperation->
select(d | d.isStatic)
Dynamic-operation-signature-set \equiv
value.base DataType.ownedOperation->
select(d | not d.isStatic)
Data-type-definition-set \equiv *value.base DataType.ownedAttribute*
Syntype-definition-set \rightarrow *isEmpty()*
Exception-definition-set \rightarrow *isEmpty()*

8.2.4.5 OBJECT

An object is mapped to an Object-data-type-definition. The remaining mapping is according to the mapping of a value, as shown in Section 8.2.4.4.

Object-data-type-definition $::$ *Sort* \equiv *object.base DataType.name*
Data-type-identifier \equiv
object.base DataType.general[0].qualifiedName
Literal-signature-set \rightarrow *isEmpty()*
Static-operation-signature-set \equiv
object.base DataType.ownedOperation->
select(d | d.isStatic)
Dynamic-operation-signature-set \equiv
object.base DataType.ownedOperation->
select(d | not d.isStatic)
Data-type-definition-set \equiv *object.base DataType.ownedAttribute*
Syntype-definition-set \rightarrow *isEmpty()*
Exception-definition-set \rightarrow *isEmpty()*

8.2.4.6 ENUMERATION

An enumeration stereotype is a collection of literals which are internally referred to as distinct values. An enumeration may optionally define specific operations on these literals. It is mapped to a Value-data-type-definition.

Value-data-type-definition $::$ *Sort* \equiv *enumeration.base Enumeration.name*
Data-type-identifier \equiv
enumeration.base Enumeration.general[0].qualifiedName
Literal-signature-set \equiv
enumeration.base Enumeration.ownedLiteral
Static-operation-signature-set \equiv
enumeration.base DataType.ownedOperation->
select(isStatic)
Dynamic-operation-signature-set \equiv
enumeration.base DataType.ownedOperation->
select(not isStatic)
Data-type-definition-set \rightarrow *isEmpty()*
Syntype-definition-set \rightarrow *isEmpty()*
Exception-definition-set \rightarrow *isEmpty()*

8.3 Example of an Implementation: an XSLT-based Approach

In this section, an eXtensible Stylesheet Language Transformations (XSLT) implementation is described for a mapping from UML to SDL design specifications. The goal of the development of a UML profile is to use several existing UML modeling tools for Internet communication protocol engineering. In fact, a few current UML tools support the XML metadata interchange (XMI) standard [OMG05e]. XMI is an XML-based standard for metamodel, metamodel and model sharing. As XMI is XML-based, an XSLT can be employed to transform XMI documents into the target language. The main benefit is that it is not required to modify a UML tool, but simply to apply an XSLT to the output of the UML tool, as shown in Figure 85. In addition, further XSLT stylesheets enable mapping to other target languages.

An XSLT stylesheet expresses a transformation. It contains rules for transforming a source tree into a target tree. This transformation is achieved by associating patterns with templates. In the source tree, elements are matched against a pattern and the template is instantiated to create a part of the target tree. The structure of the target tree may be completely different from the source tree's structure. By means of a stylesheet, elements from the source tree can be re-arranged, exchanged and arbitrary structure can be added.

To be more precise, a stylesheet contains a set of template rules. A template consists of two parts: a pattern which is matched against the nodes in the source tree as well as a template which can be instantiated to form a part of the target tree. Each template is instantiated for a particular source element to create parts of the target tree. A template can also specify literal result element structures or it may also specify instructions from the namespace to create target tree fragments. If a template is instantiated, each instruction is executed and replaced by the target tree fragment that it creates. Descendant source elements can be selected by further instructions. Processing of descendant elements creates a target tree fragment by finding the applicable template rule and instantiating its associated template. When started, the target tree is constructed by finding the template rule for the root node and instantiating its template. XSLT uses the XPath language [CD99] to browse the source tree while providing additional functions in order to add flexibility to XSLT.

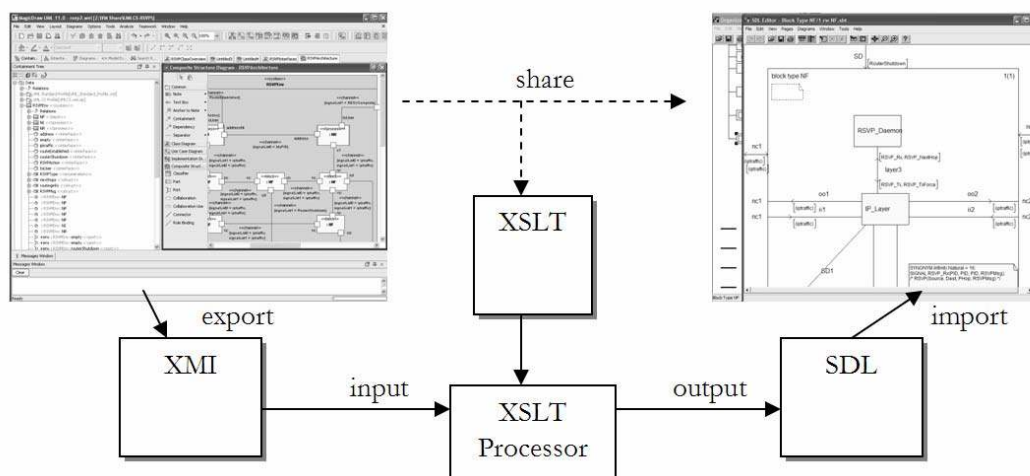


Figure 85: Using XSLT Principle

Several XSLT processors like Xalan [Xal] or Saxon [Kay06] are available for the mapping process. XPath and XSLT belong to the eXtensible Stylesheet Language (XSL) family of languages that describe how files encoded in the XML standard can be transformed. The complete XSLT stylesheet can be found in the Appendix B: XSLT Stylesheet for UML CS. This stylesheet has been created with respect to the OCL mapping specification presented in Section 8.2. However, this mapping is only manually derived and it differs in some points for reasons of practicability and development. It has also to be noted that some elements are not mapped because they are not bound to their owning or associated element within the XMI representation. This can be completed by using a future version of the modeling tool that resolves this issue.

The XSLT stylesheet parses an XML document containing a UML 2 diagram structure described in an XML Metadata Interchange format (XMI). The output of the parsed stylesheet is a valid SDL-96 CIF level 0 (formerly SDL/PR, the phrase representation for SDL) that can be used for prototyping purposes in any SDL modeling application. By using XSLT, the original XML document tree is browsed and translated into its equivalent SDL representation.

The XSLT-stylesheet allows to define of debug-flags which result in a verbose output of the mapping process. Before the mapping process starts the XMI version is checked first by the `@xmi:version='2.1'` rule. This assures XMI version 2.1 compliance. After this general source tree compatibility check, the XMI document is parsed starting from the root node. From this node the tree is traversed according to the SDL logic. The path of the process is followed by resolving the appropriate transitions. Activities are converted to procedures and high-level model elements are converted to a pre-defined set of SDL statements. Parts of the stylesheet are included from additional files to retain clearness and flexibility. The applied stereotype is determined by a sub-node of type `<appliedStereotypeInstance>`. Within this node, there is a further entry `<referenceExtension>` that identifies the appropriate stereotype extension. For instance, the system agent is encapsulated by the node of type `uml:Class` and defines a reference to the stereotype extension named `UML CS Profile::system`. This is where the first template rules match on. Within the system agent, there are several sub-nodes within the `<nestedClassifier>` attribute of various types like: `uml:Signal`, `uml:DataType`, `uml:Enumeration` and `uml:Interface`. By means of a choose-rule, the type is determined and results in the output of different templates for the SDL/PR target. Processes are created if a nestedClassifier of type Class and stereotype process is found. First, the signal routes are created, after that the variable definitions and procedure references will be created. Originating from the start state, the process state machine is generated. All outgoing transitions are evaluated and finalized by the `nextstate` expression. If a trigger is defined a different template is used. Then, all outgoing transition actions are created from each state. Data types are matched by special template rules according to their specified type. As the available data types are inherited from SDL a one-to-one mapping is sufficient.

The XSLT stylesheet successfully maps a given UML model to an SDL/PR based specification. While simple constructs (such as states, assignments, agents) can easily be mapped to SDL, considerable effort is required to map high-level and more complex specification. In fact, this has not been achieved with the stylesheet. The XSLT approach seems to be somewhat unsuited for the mapping of constructs that result in complex target constructs. For instance, the UML CS profile allows defining a process within a system. In SDL-96, this is not allowed, because the process must be contained in a block first. To generate an implicit block by means of the XSLT requires significant more effort as the connections (channels) and gates have to be derived from the connection between the process and the system agent. This also applies to the dynamic port concept that is a new language feature of this profile. In addition, soft states, which require implicit timer declaration and management, have not been implemented. This is also caused by the limited support of passing variables as parameters between stylesheet templates. These problems are very intricate to solve, and it has not been achieved

to implement these complex features in a full-fledged manner. This imposes a huge set of stylesheet parts that make maintainability of the overall stylesheet very difficult. It is proposed to replace the XSLT-based implementation by Java-Document Object Model (DOM) technology in the future. Nevertheless, the XSLT-based implementation demonstrates the soundness of the overall profile's concept and serves as a basis for further development.

8.4 Summary

In this chapter, a mapping from the UML CS profile stereotypes to SDL has been presented. First, a mapping of the UML CS modeling elements has been defined by means of the Object Constraint Language (OCL). This has covered all structural and behavioral model elements provided by the UML CS profile. Second, a concrete implementation based on the OCL-based mapping rules has been developed.

The first section has introduced the mapping process by means of OCL and has defined the type conversion rules and metamodel creation from the abstract syntax (AS1) tree of an SDL specification. This is a prerequisite for a mapping constrained by the formal specification language. In the second section, the extended model elements have been mapped to an SDL element defined in the AS1. This mapping is specified by OCL expressions. The mapping to AS1 was chosen to enable UML tool vendors to generate an abstract syntax tree from UML CS specifications directly. This eliminates the need to derive it from the concrete syntax. However, some concepts were not mapped to SDL, as there is no equivalent language construct available. In addition, a concrete mapping has been developed by means of an eXtensible Stylesheet Transformation (XSLT) in the third section. This stylesheet allows processing a UML CS model that is available in the XML diagram interchange (XMI) format. It is able to generate a complete structural and behavioral SDL-96 specification. The stylesheet's templates and template rules are based on the OCL mapping defined in the previous section. The concrete mapping to SDL-96 was chosen because SDL-96 is well supported by tool vendors. Moreover, several different compilers are available. In contrast, SDL-2000 is currently not sufficiently supported by commercial tools. Unfortunately, the XSLT approach was not the optimal way to implement the mapping. Generation of implicit and complex constructs requires significant effort for the development of XSLT-stylesheets. It is considered to replace this approach in the future.

However, the mapping to SDL-96 has been manually implemented with respect to the OCL mapping rules. No automatic checking of the OCL constraints within this implementation has been done. This is part of the future work, so that the formal mapping constraints can be used to validate a mapping to SDL automatically.

9 Case Study

This chapter describes an Internet signaling protocol specification – the Resource Reservation Protocol – by using the UML CS profile described in this thesis. This model is almost identical to the model that has already been discussed in Chapter 5. The only differences can be found in the application of some of the new language constructs of this profile for Internet protocols. This chapter only focuses on the most interesting parts of the model.

The UML modeling tool used for this purpose does not visualize all properties of elements that are defined in the model repository. The main stereotypes are explicitly turned visible to outline the various extended elements. However, elements such as ports do not show their stereotype extension although they have one stereotype extension assigned. This is an internal restriction of the used modeling tool. However, this imposes nothing on the actual model stored in the model repository having the required stereotypes applied. Also note, the used modeling tool is not fully compliant as required by this profile. To the best of knowledge, there is no UML modeling tool on the market at the time of writing that reaches the required UML compliance level three capabilities. Nevertheless, to overcome this limitation some stereotypes have been adapted to enable the application of this profile. These adapted stereotypes have a leading X added in order to depict this.

9.1 Architecture

First, the following UML diagrams give an overview of the RSVP system specification and description in the UML 2 with the UML CS profile applied. The class diagrams show the architectural model elements that are part of the system. However, the used modeling tool, MagicDraw 11 [Mag], does not show all intimate details of the model repository in a diagram. Figure 86 depicts the agents, their inner agents and attributes. The outermost agent is the system *RSVPEnv*. This describes the system and denotes the agents that are part of the system. For *RSVPEnv*, there are anonymous instances shown of a process *NI*, a process *NR* and five instances of the block *NF*. The declaration of the process *NI* and *NR* is also shown with several attributes of local scope. These processes are active classes indicating that they execute a behavior after their instantiation. The block *NF* defines constant values of a specific type, indicated by the stereotype *constant*. These values are marked as being read-only and cannot be changed during the execution of the system.

As shown, the block *NF* defines another nested block *IP_Layer* which is managing the packet routing functionality and a process *RSVP_daemon* which manages the various RSVP signaling states. The block *IP_Layer* comprises two processes: The *Routing* process and the *Forwarding* process. While the first process determines the next hop to which a received datagram packet has to be forwarded, the latter process actually exchanges datagram packets with its environment.

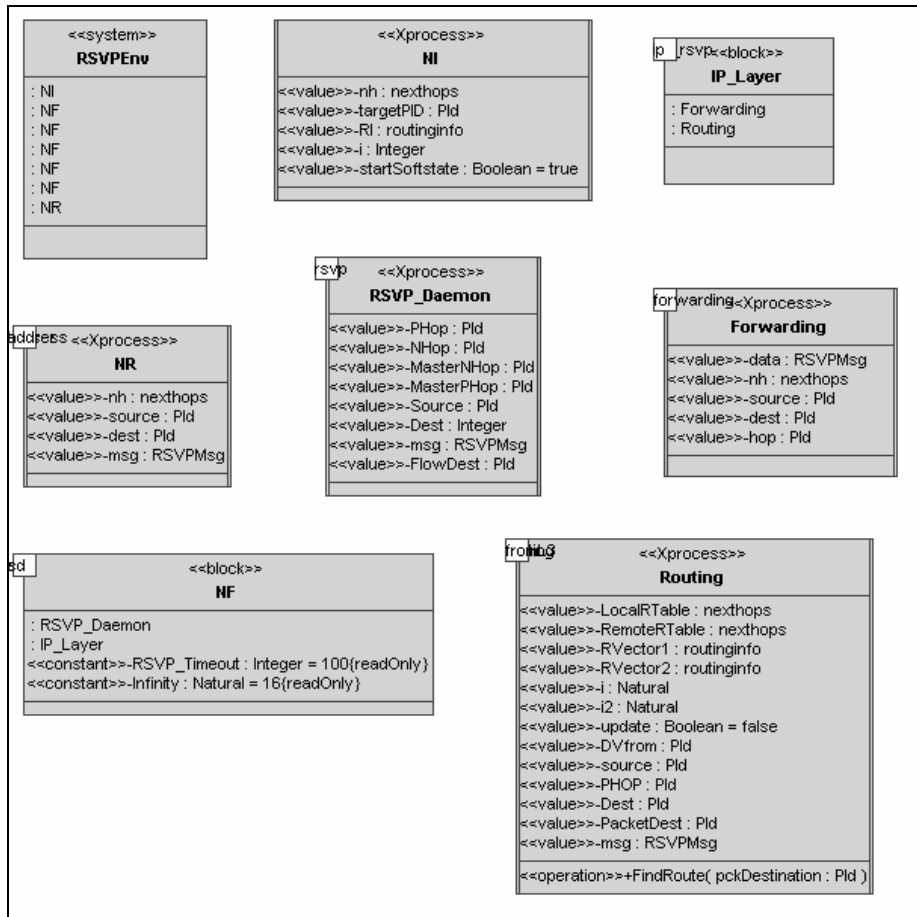


Figure 86: RSVP Agent types

The next Figure 87 defines the signals that are used by the system. Some of the signals define attributes of various types. The signals within the scope of the system are listed with their attributes if any. In addition to the signal definitions, the signal lists are indicated with the stereotype *signalList*. For instance, in the signal list *iptraffic* it is shown that the signals *DistanceVector*, *datagram*, *LinkFailure* and *ForceDVUpdate* are subsumed under one single identifier. This is used as shorthand notation to specify signal lists for channels.

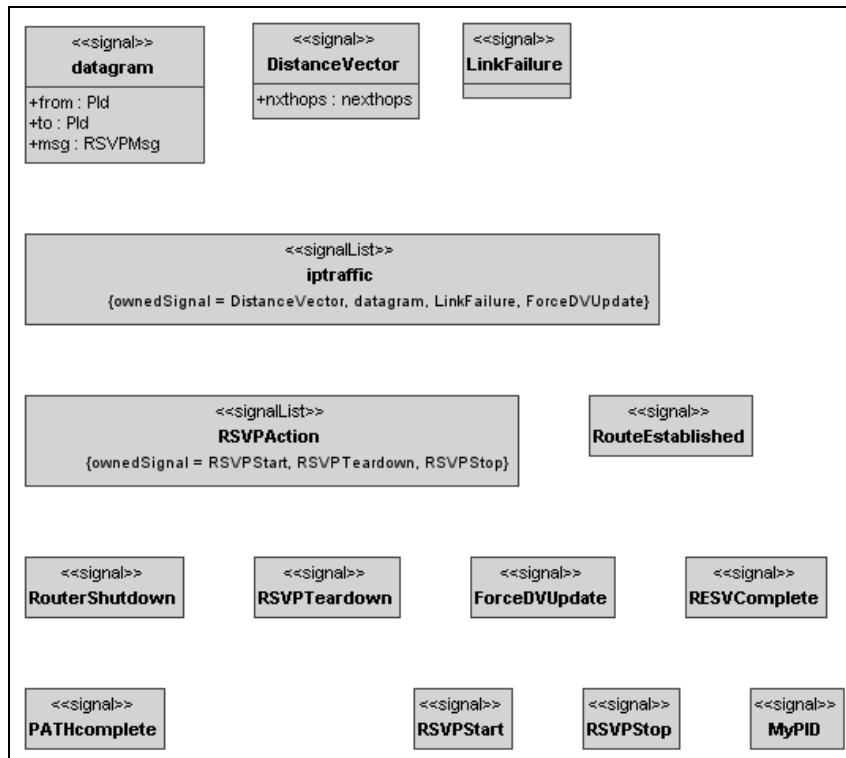


Figure 87: RSVP Signal and Signallist Definition

The class diagram also shows the enumeration type *RSVPType*. This enumeration consists of the literals *Path*, *Resv*, *PathTear*, *ResvTear*, *PathErr* and *ResvErr*. These literals indicate the type of RSVP message that is sent between the nodes.

Figure 88 provides a view on the composite data type definition, enumerations and interfaces. The enumeration *RSVPType* defines some of the various RSVP message types which are required for signaling purposes. The structure *nextops* is used for exchanging routing information between hosts by means of the signal *DistanceVector* shown above. The structure *routinginfo* is used to store a table of the neighboring hosts by each node. The interfaces define signals and operations that can be conveyed through this interface. However, the direction of the signals is determined by their association to the specific port instance (provided or required).

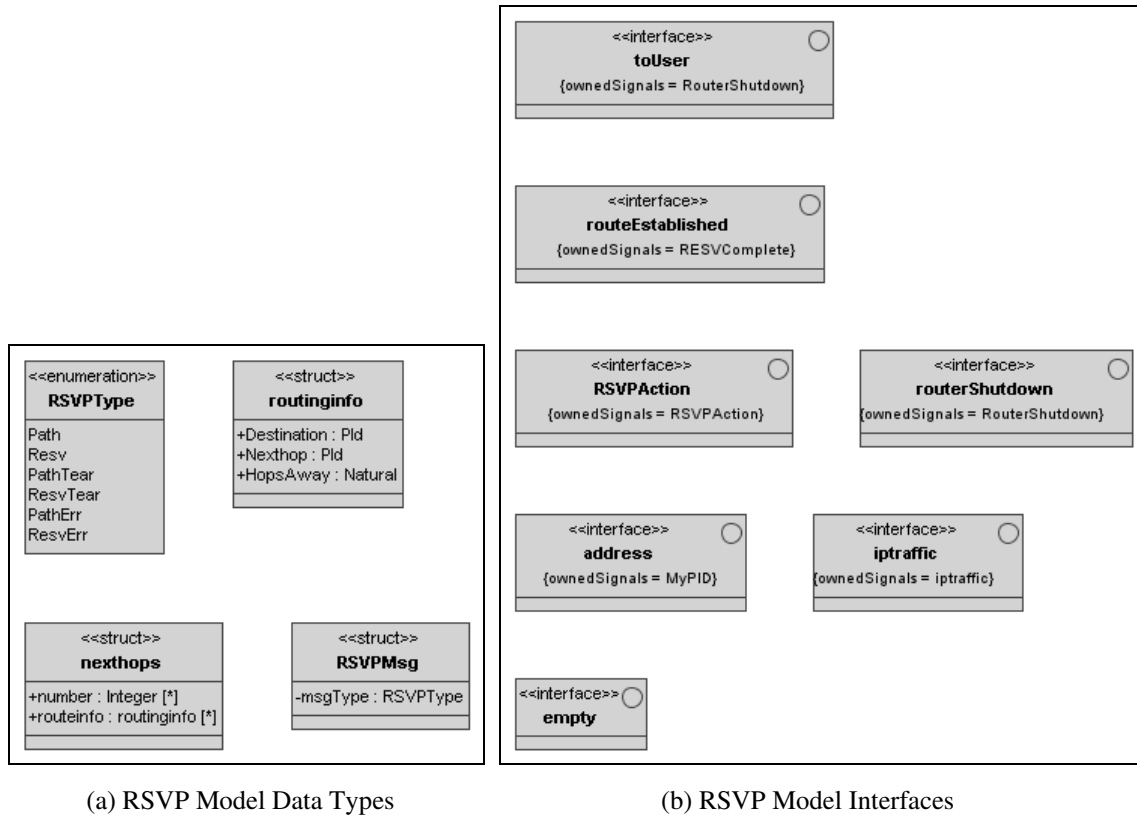


Figure 88: RSVP Data types and Interfaces

The composition of the system is shown in the Figure 89. This system shows two anonymous (unnamed) processes instantiated from the process type *NI* and *NR*. Additionally, there are five unnamed instances from the block type *NI* present. Analogously to the description in Section 5.1, all agents automatically initiate their communication.

The channels inbetween the *NI*, *NF* and *NR* nodes convey the signals defined in the signal list *iptraffic*. These signals are the *datagram* signal which itself forms the datagram packet in order to convey some information. The signals *ForceDVupdate* and *DistanceVector* are both routing related signals triggering routing table updates between nodes. The *LinkFailure* signal is used to indicate that a node is shutting down. This signal is then sent to all neighbors triggering some behavior.

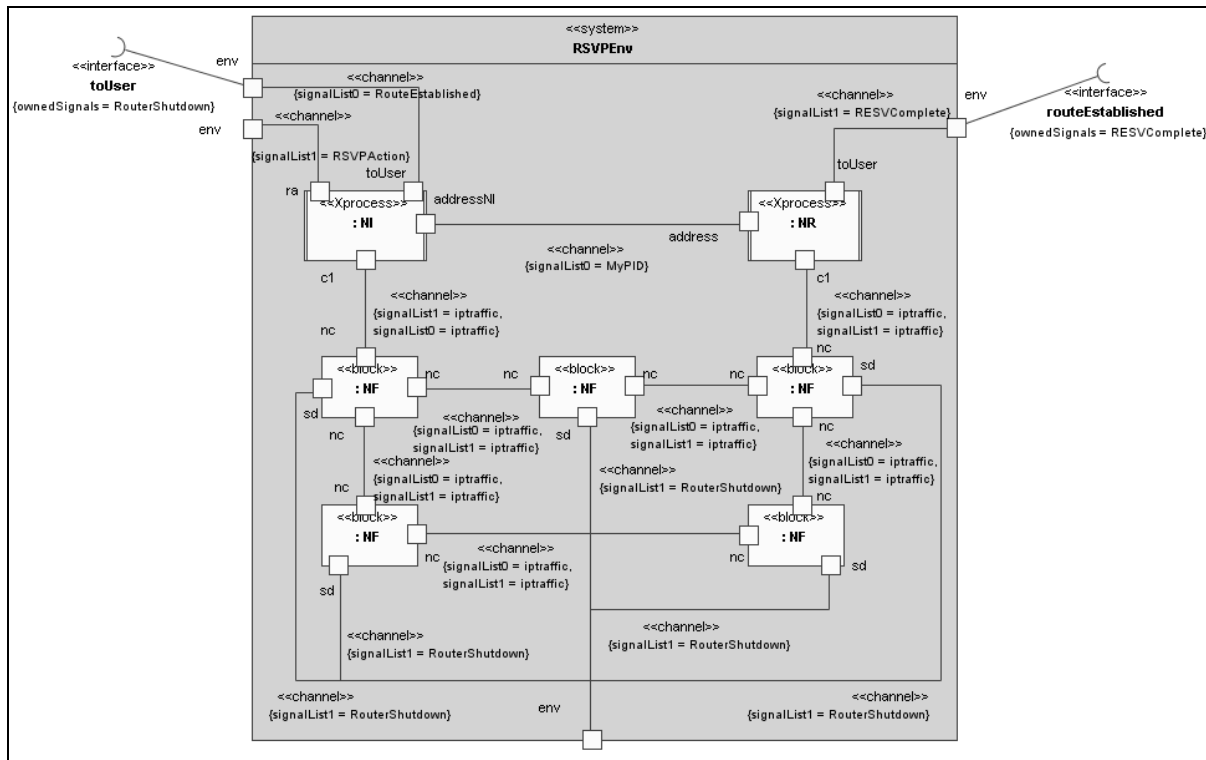
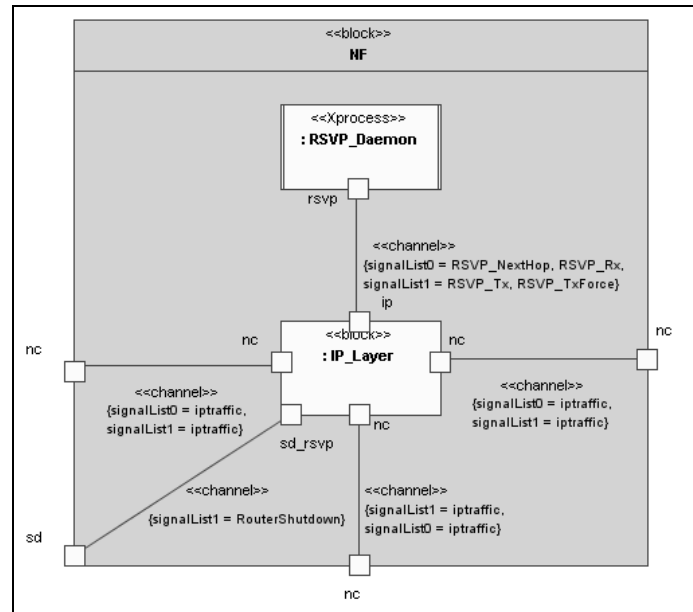


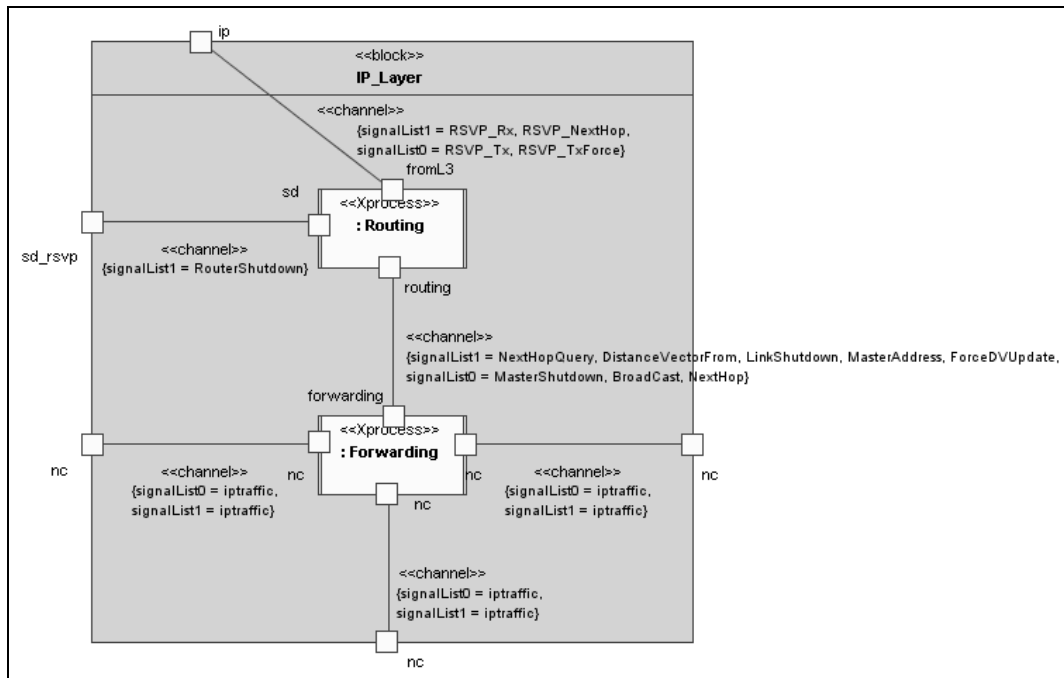
Figure 89: UML CS System of RSVP Model

When the system's execution is starting, the *NR* announces its process identification to the *NI* by sending the *myPID* signal. The *NF* instances start to send routing table information to their neighbors. This information is used to build up the internal routing tables within each block. When the environment triggers the *NI* the *NI* initiates the RSVP path reservation by sending a RSVP message to the *NR* via a path of several *NF* instances. The *NF* instances forward the received message to the corresponding neighbor based on their internal routing table information.

This system also defines the use of the dynamic ports applied to the *NF* nodes. The *NF* block type attaches three dynamic ports grouped together by their name *nc*. While the *NF* nodes being directly connected to *NI* and *NR* have all three dynamic ports attached, the remaining *NF* nodes use only two dynamic ports. The channels between the ports define the signals that can be conveyed to the endpoints. The correlation between such a *signalList* and the endpoint can only be made by the evaluation of the ordered list of endpoints. The ports of the system enable the communication with the environment. Signals which are accepted from the environment or can be sent to the environment are specified on their corresponding provided and required interface definitions. They are not visible in this diagram.



(a) Composite Structure of the NF block



(b) Composite Structure of the IP_Layer block (within the NF block)

Figure 90: UML CS Overview of NF block

The composite structure diagrams in Figure 90 show the owned agents of the *NF* block type. As already discussed for the class diagrams, this is the *RSVP_Daemon* process and the *IP_Layer* block. The latter block decomposes into a *Routing* process and *Forwarding* process. Within the *NF* block all dynamic ports *nc* of the *IP_Layer* block are connected to the *NF* block's dynamic ports.

The *IP_Layer* is the responsible entity for datagram packet routing. A RSVP datagram is received and forwarded to the *RSVP_Daemon* process. It processes this message and may create some other datagram packets to be forwarded. The signals *RSVP_NextHop* provide the Pid of the next hop node towards the datagrams destination. The *RSVP_Rx* signal conveys the RSVP message type. The signals *RSVP_Tx* and *RSVP_TxFoece* trigger the creation of an RSVP datagram packet with the appropriate

attributes. The neighboring hop, to which this datagram has to be sent, is selected by the *IP_Layer* by the signal *RSVP_Tx*. In contrast, the next hop can explicitly be selected using the signal *RSVP_TxForce*. The *RouterShutdown* signal indicates to the *IP_Layer* that the node has to be shut down.

9.2 Behavior

The following section describes the behavior UML model with the UML CS profile applied. However, the diagrams themselves do not represent all descriptions that are contained in the model repository. Therefore, some associations cannot be recognized by the diagrams' representation only. For example, there is no direct visible link between an activity name on a state machine transition and the actual activity. Its name is not part of the diagram.

9.2.1 NI

The behavior of the *NI* process is modeled by means of a state machine, shown in Figure 91. Beginning with the start node, the *NI* initiates the execution of the RSVP signaling process. When the *NI* process starts, it first initializes its variables by means of the activity *init*. Then it enters the state *idle*. From this state the *NI* waits for the *MyPID* signal. This signal is used to determine the *Pid* which represents the Internet address of the target host. The *NI* saves this value within the *getDestIP* activity. Hence, the *NI* waits for a signal event from the environment triggering the RSVP path state set-up. This is activated by the signal *RSVPStart*. The activity *startSS* changes the Boolean variable *startSoftstate* to a true value, thereby enabling a soft state trigger at the *idle* state. The soft state is now set with the current time adding 10 time units. After time-out, this triggers a new RSVP state refresh message to the RSVP path down to the *NR*.

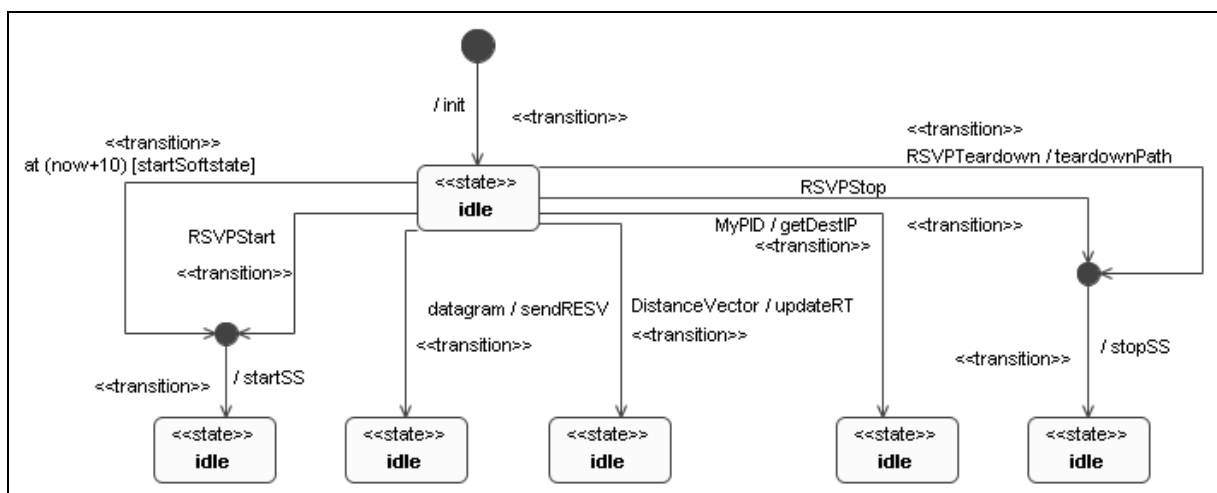


Figure 91: Statemachine of NI Process

On the left of Figure 92 the *init* activity is shown. The task assigns an initial tuple value of *self, self, 0* to the *nh* array zero index. The *nh* array represents the routing table and the destination of a packet, *self*, the neighboring host to which the packet has to be forwarded in this case – of course to *self* – and the distance in intermediate hops – of course zero.

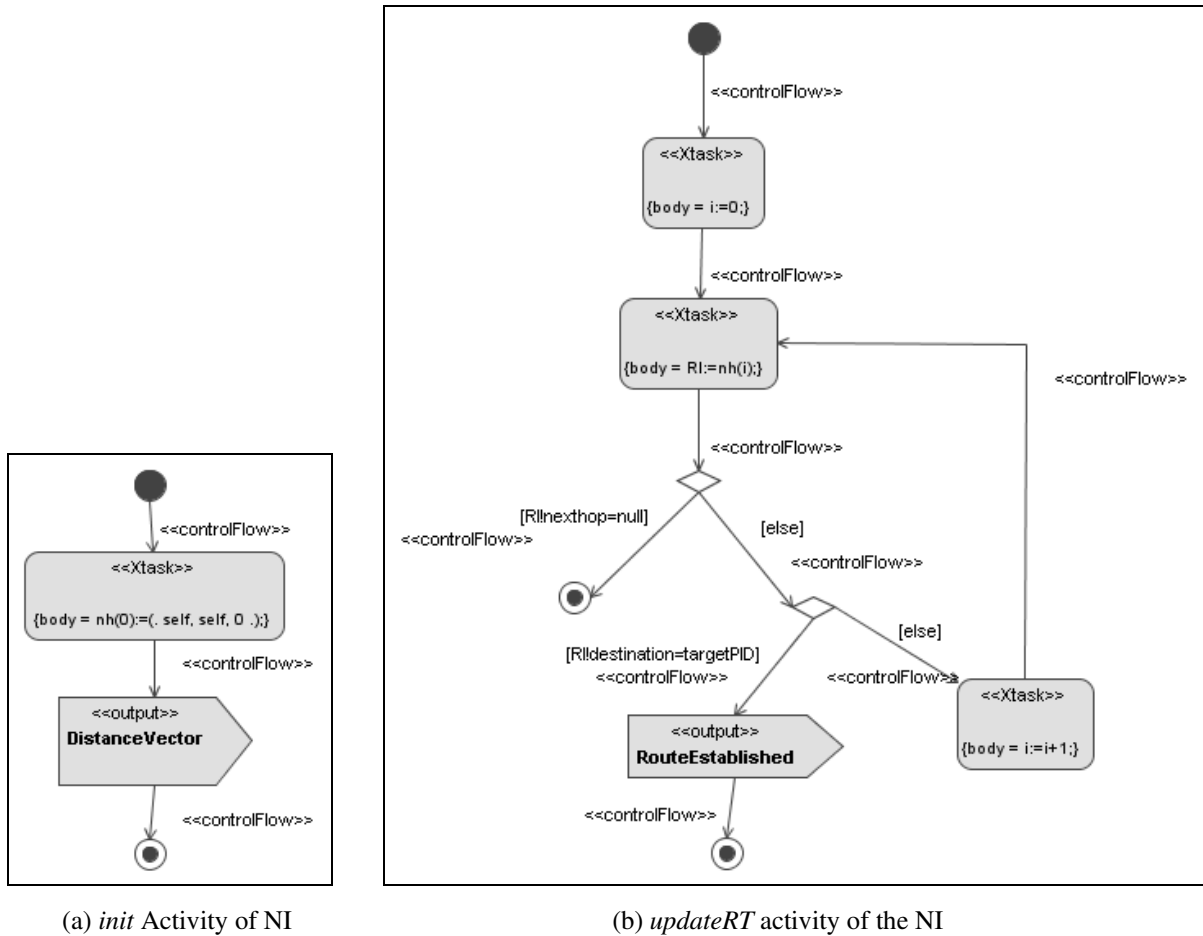


Figure 92: Activity Diagrams *init* and *updateRT* of NI Process.

On the right is the *updateRT* activity. This activity checks if the routing information received from its neighboring NF instance already contains routing information about the target process NR. If this is present, the activity will send the *RouteEstablished* signal to the environment. This indicates that the routing tables have converged to a stable state and have enough information to process packet routing successfully.

9.2.2 NR

The NR process is the target of the RSVP path reservation. Figure 93 shows the state machine definition of the NR process. The process initializes by executing the *init* activity. Then, the state *idle* is entered. Being in the *idle* state the process waits for a *datagram* signal or for a *DistanceVector* signal event which is simply discarded.

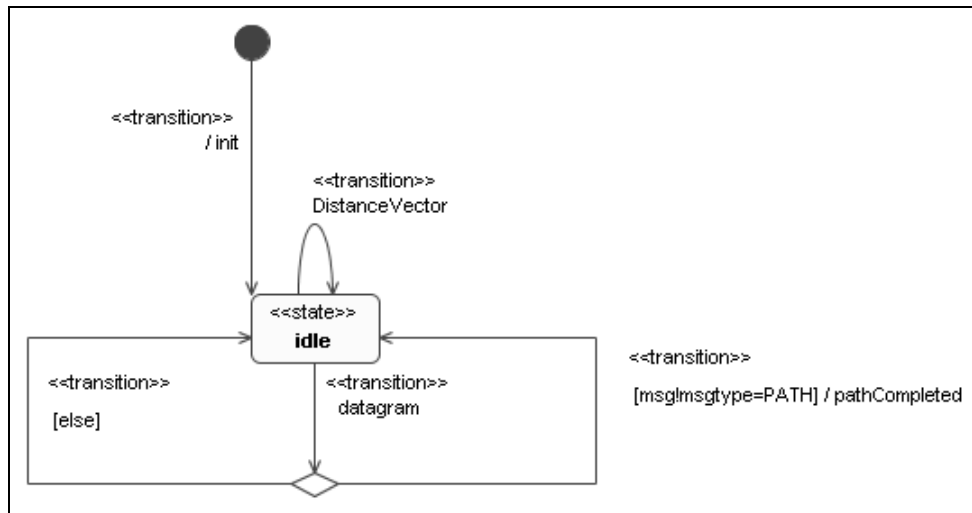
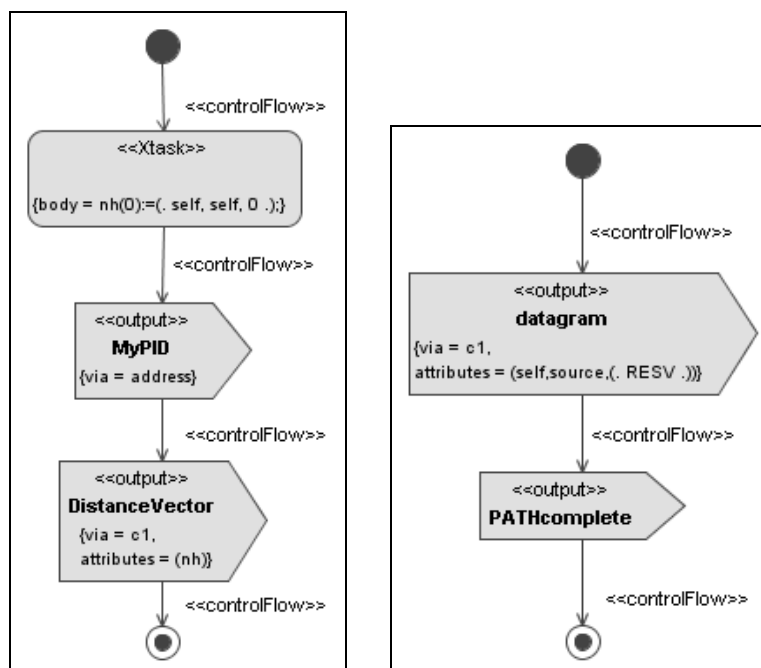


Figure 93: State Machine of NR

When the datagram signal is received, the *NR* evaluates the *msgtype* attribute of the received datagram signal. The assignment specification of the *datagram* signal event is not visible in this diagram. Nevertheless, this assignment is defined in the model. If the *msgtype* is equal *PATH*, the *pathCompleted* activity is executed returning a datagram signal with *RESV* message type and a *PATHcomplete* signal. This is shown in the following activity diagrams in Figure 94.

(a) *init* Activity of the NR process (b) *pathCompleted* Activity of the NR processFigure 94: Activities *init* and *pathCompleted* of NR

9.2.3 NF

The *NF* behavior is separated into a routing and a RSVP entity. The RSVP is a process and its behavior is discussed first. As shown in Figure 95 and Figure 96, the RSVP process starts up and waits for the reception of a *PATH* message signal. Otherwise, it will result in a *RESVError* response. The RSVP process forwards the *RESV* message and enters the *WAIT_RESV* state.

From this point, RSVP waits for the *RESV* message to arrive upstream. If it does not arrive in time or a *PATHTEAR* message is received, it will return to the *idle* state. Another *PATH* message will keep the process in the *WAIT_RESV* state. If a *RESV* message is received, which completes the downstream path reservation procedure, the soft state of the RSVP state is set up and the *RESV_mode* is entered. Being in this state indicates a successfully completed reservation. Roughly speaking, the remaining behavior consists of waiting for in time refreshes from the *NI* and from the *NR*. The messages received are forwarded. If a time-out of the refresh cycles occurs, the RSVP process falls back to the *idle* state.

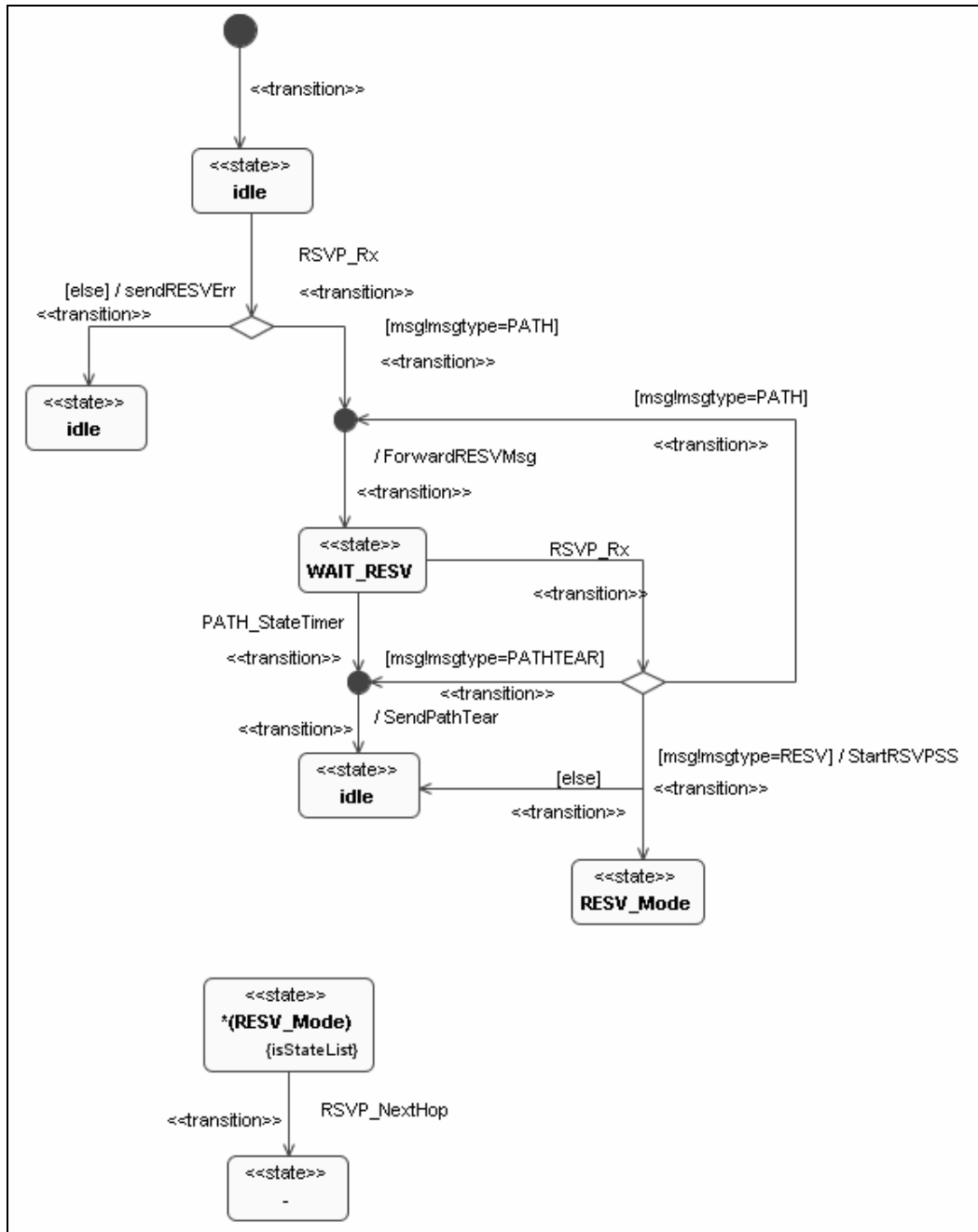


Figure 95: First Part of State Machine of RSVP process

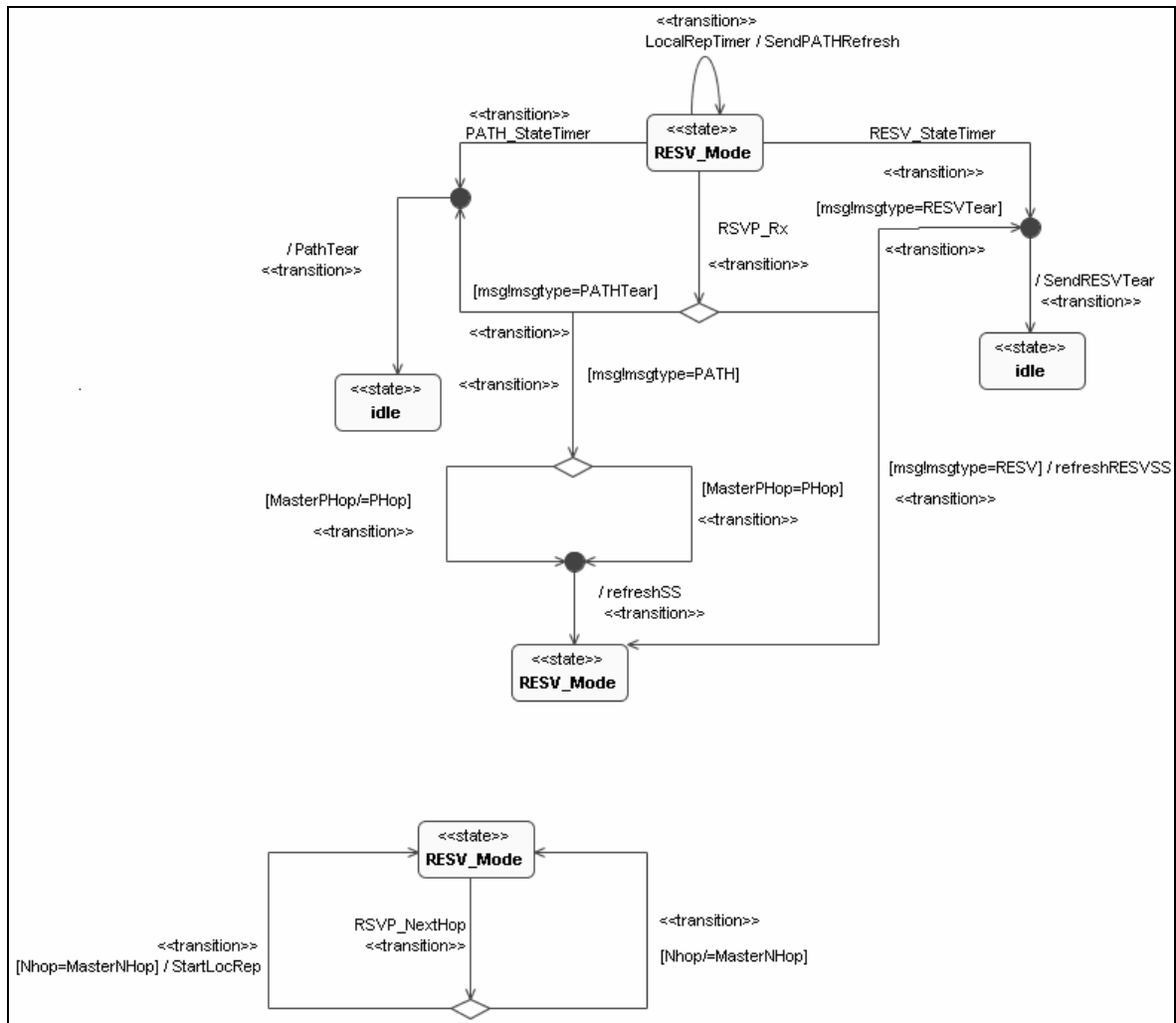


Figure 96: Second Part of State Machine of RSVP process

The remaining entity is the routing layer. The routing layer facilitates message reception, interface selection and message sending. The routing layer consists of the forwarding process which is the entity forwarding a message through a specified interface and the routing process that selects the outbound interface.

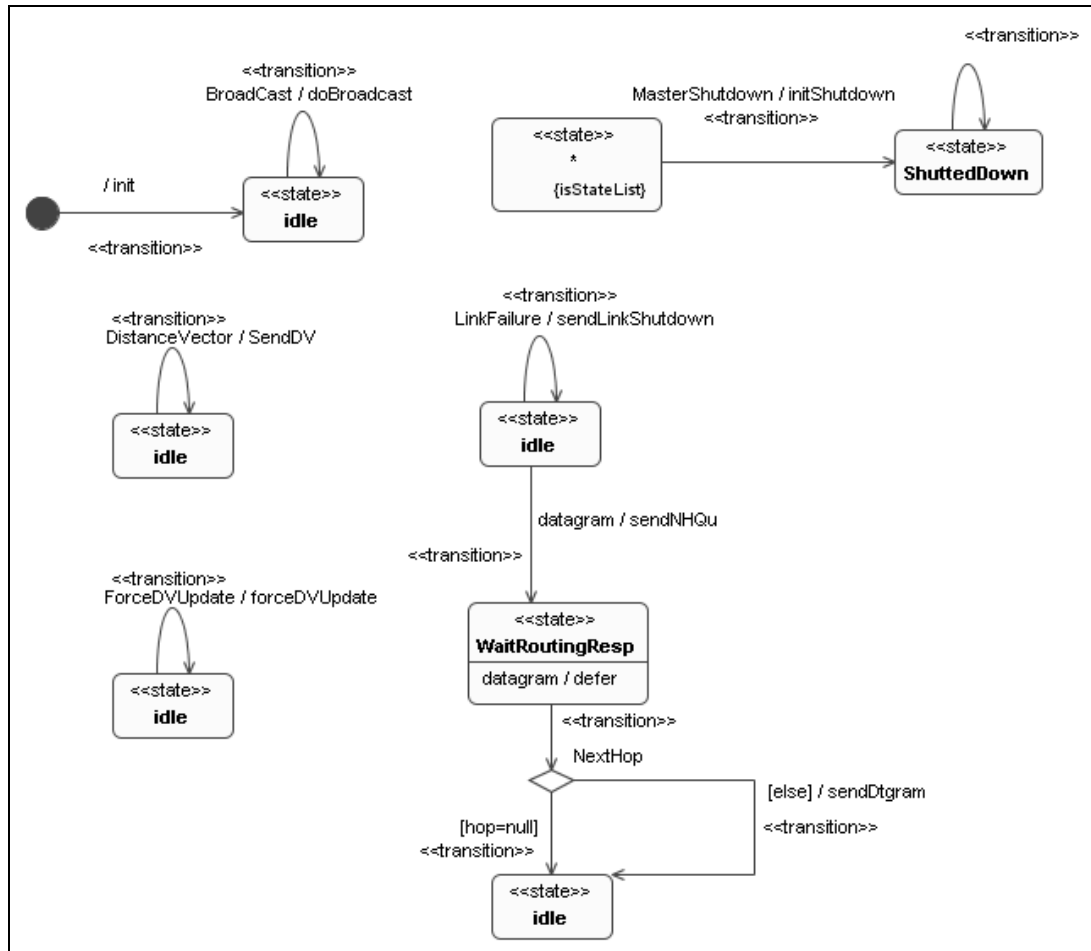


Figure 97: State Machine of *Forwarding Process*

As shown in Figure 97, the forwarding process first initializes and enters the *idle* state. In any state, the reception of a *MasterShutdown* signal initiates the shutdown sequence by executing the activity *initShutdown* and enters the *ShuttedDown* state. From this point, any signal event is discarded and the state is never left.

When the *idle* state is active, the state machine waits for several types of signals. The *distanceVector* signal event and the *ForceDVUpdate* are forwarded to the routing process. If a datagram is received, a signal is sent to the routing process querying the next hop's *Pid* to which the datagram is to be forwarded. The process enters the *WaitRoutingResp* state and saves any datagram signal for deferred processing that is received. If a response signal is received from the routing layer constituting a valid *Pid* the datagram is forward to this node. If the *Pid* is a null value the pending signal is discarded.

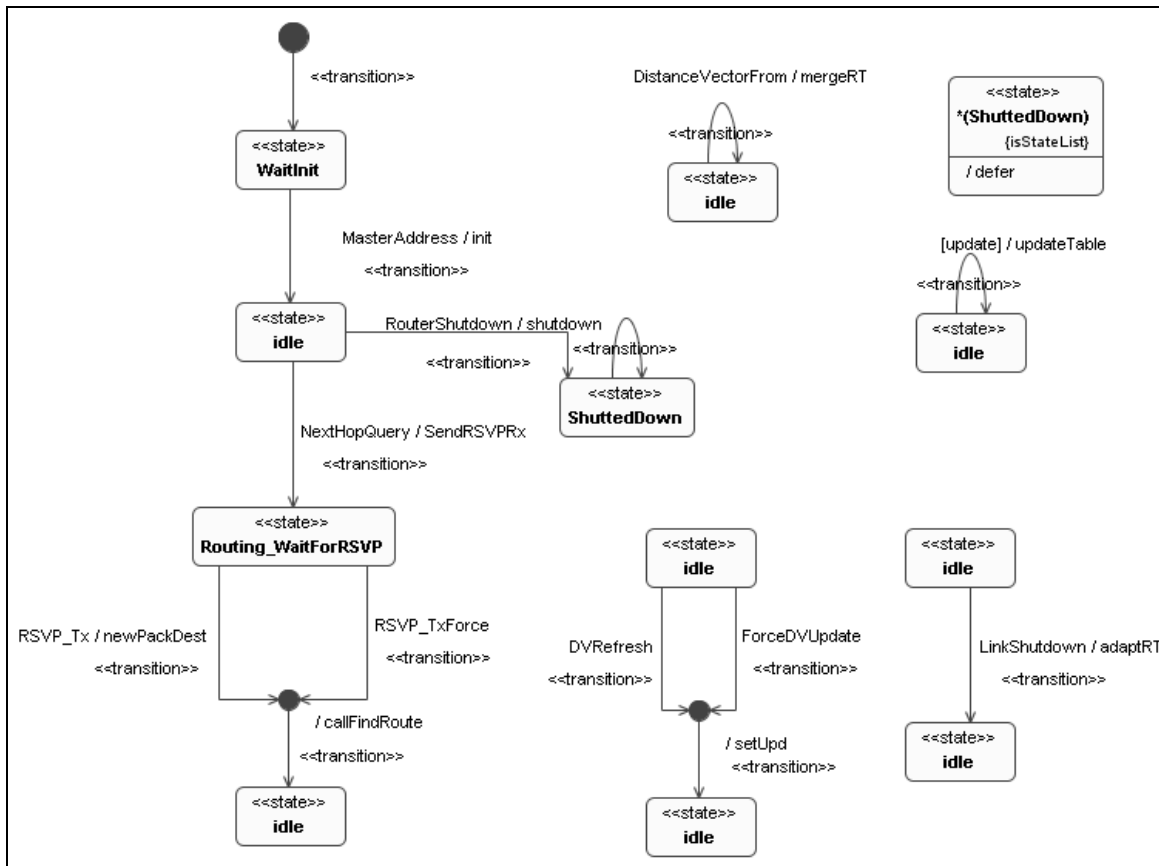


Figure 98: State Machine of *Routing* Process

The state machine of the *Routing* process is shown in the above Figure 98. This process facilitates the next hop selection. The process starts by entering the *WaitInit* state and expects a *MasterAddress* signal event that assigns a unique *Pid* (which is treated as an IP address). Then the process executes the initializing *init* activity. Note that with the exception of the *ShuttedDown* state, all triggers are deferred. The routing process waits for a *NextHopQuery* signal and forwards this query to the RSVP process if it is an RSVP message type. When a response signal is received, namely the *RSVP_Tx* or *RSVP_TxForce* signals, the *callFindRoute* activity is executed that invokes another operation. This operation evaluates the routing table for a next hop candidate. The result is then reported to the *forwarding* process. Figure 99 displays these activity diagrams.

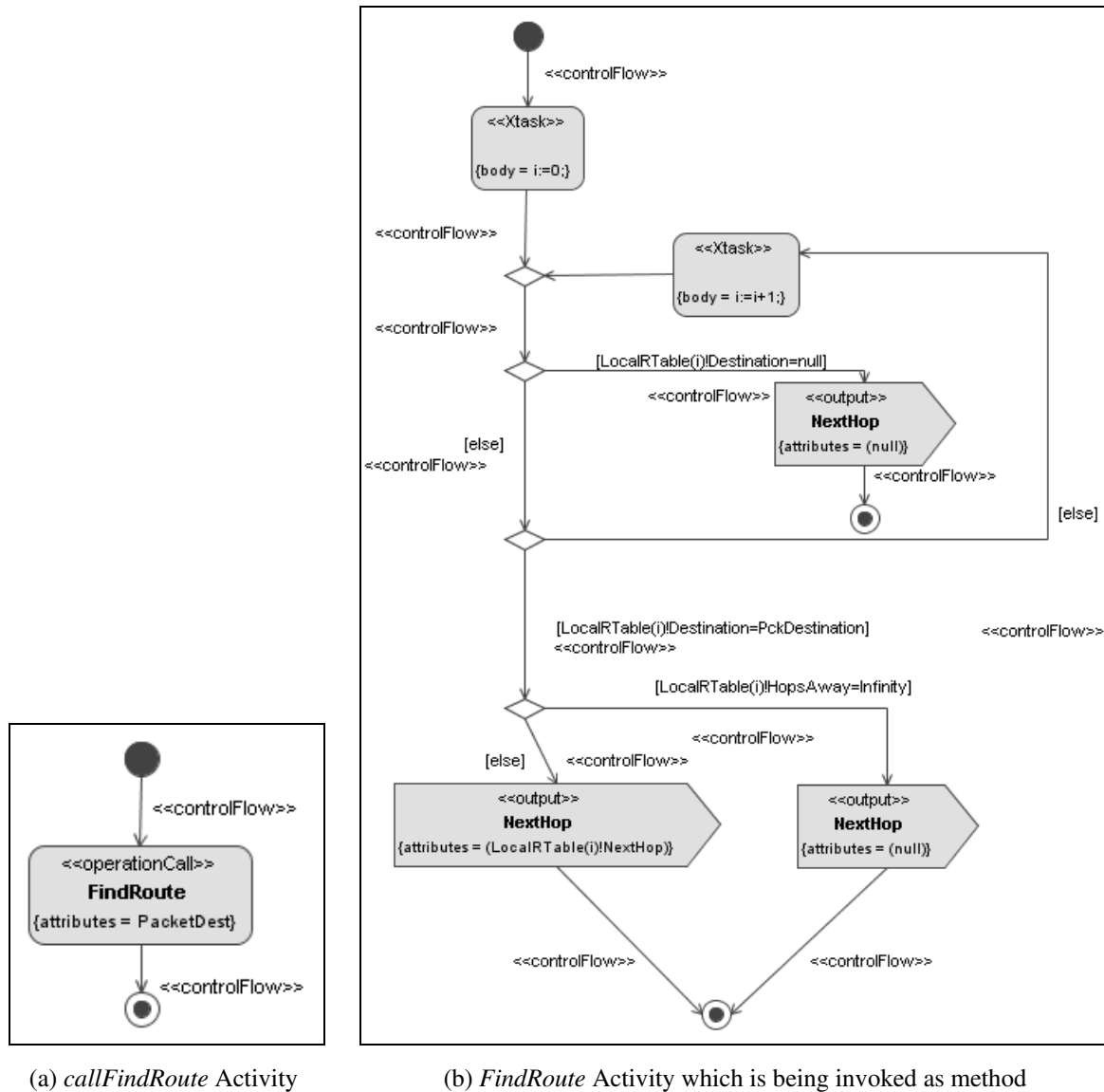


Figure 99: *callFindRoute* and *FindRoute* Method Activity

In addition, the routing process uses several signals to update and to process modifications to its routing table. *DVRefresh* and *ForceDVRefresh* trigger broadcasts of a routing table refresh message to neighbors. The *LinkShutdown* signal event notifies the node that the neighbor has been shut down. Therefore, the *adaptRT* activity deletes the routing table entry of this node. The *DistanceVectorFrom* signal event conveys routing table information from a neighboring node. This information is used to merge the local routing table with the updates of the network topology information.

9.3 Summary

In this chapter, a feasibility case study for modeling Internet signaling protocols – the Resource Reservation Protocol (RSVP) – by means of the UML CS profile has been presented. Some of the advanced modeling features have been used which were discussed in Section 5.2. The first section has presented a brief description of the architectural diagrams of the system model. This included the description of structural features of the agents, such as signal, variable declaration and the composite structure. In addition, the definition of signal channels with the appropriate signal lists and interfaces

with their owning ports were discussed. The focus was on how a static structure of such a system model can be described.

The second section has described the behavior of the various agents of the system. There are two processes that are responsible for initiation (*NI*) and responses (*NR*) to a specific reservation. In addition, there are several instances from a forwarding agent type (*NF*). This node is responsible for forwarding and routing datagram messages between the *NI* and *NR*. The behavior of all processes has been described by use of state machines and activities, although not all activities were shown.

The benefits of the use of the profile (compared to the SDL-based model) are that the connections between the various processes do not require to introduce dummy processes. Each agent can connect to any amount of other agents. This eliminates the need to define a new agent type for each different amount of ports. The soft-state concept is now integral part of a state and does not require to use the timer mechanism explicitly. This provides a means to avoid low-level mechanisms for time-related states. Although it has not been specified within this model, the random functionality could be used to define router shutdowns in a pure random manner. In this model, this shutdown is still triggered by a specific signal received from the environment.

10 Conclusions and Outlook

In this thesis, a new Unified Modeling Language 2 (UML) Profile for Communicating Systems (UML CS) has been described and defined. This profile tailors the UML 2 for the unambiguous specification and description of communication and signaling protocols for the Internet. This profile enables analysis, simulation, validation and generation of an implementation of UML-based protocol specifications. The profile is driven by the concepts and expressiveness of the Specification and Description Language (SDL). The goal of this profile is to bridge the gap between the requirement and analysis stage and the design stage by combining the strengths of the UML and SDL. While the UML features multiple viewpoints on same system, informal object models and property model views, SDL offers detailed formalized object models with respect to execution semantics.

For the language concept of the profile, an analysis of SDL with respect to its suitability for the specification and description of Internet signaling and communication protocols has been presented. For this purpose, an Internet signaling protocol – namely the Resource Reservation Protocol (RSVP) – has been specified with respect to routing and intermediate node failures. This model has driven the analysis on shortcomings of SDL. The findings of this analysis were that SDL lacks improved support for randomness, a gate or sender specific signal consumption, dynamic gates for network topology modeling and soft states. These features have facilitated additional high-level concepts in the UML CS profile for communicating protocol engineering especially for packet switched networks.

The UML CS profile definition is based on the following steps: The stereotypes which extend the UML metaclasses and compose this profile package have been carefully adapted to accomplish the task. Formal constraints of the stereotypes have been defined for the abstract syntax and static semantics. For the semantics of the profile, a mapping specification has been defined by means of the formal Object Constraint Language (OCL). This specification provides the necessary rules to enable a distinct mapping from a UML CS model to an SDL system specification according to the defined semantics. Given this mapping specification, an implementation has been developed to demonstrate the feasibility and soundness of the profile's concept. This implementation successfully maps a UML CS model to a complete architectural and behavioral SDL system specification. This mapping has been implemented by using the eXtensible Stylesheet Transformation Language (XSLT). The implementation processes an XML metadata interchange (XMI) file output of a UML modeling tool. A case study has been presented to show how an RSVP specification, which has already been used for the analysis of SDL, can be specified using this profile. Some of the new language features have been used in this case study.

In summary, it can be stated that this profile is the first UML 2-based profile that defines a language especially for the specification and description of Internet communication and signaling protocols. In addition, it is the first UML 2-based profile that defines a mapping from UML 2 design specifications to SDL. It also takes into consideration the changes of the upcoming UML revision 2.1 draft, because it already contains several error corrections compared to the current UML version 2.0 in-force. It can further be stated that this UML profile is currently the only one that uses a formal language in order to define constraints of the stereotypes and for a mapping specification to SDL-2000. This formality is very beneficial for automatic validation of a model and its mapping. This can enable tools automatically to check the partial correctness of UML CS models and their mapping to an SDL specification. As this profile is not limited to SDL, it also includes several extension points that are incorporated to enable a mapping to other formal description techniques with respect to the semantics provided by the SDL mapping rules.

The future work of this profile is to incorporate the language features of SDL-2000 which are currently not supported by this profile. This includes the support for agent templates and exceptions. The implementation is still in a prototype stage and was mainly used to show soundness of the profile's concept. Robustness aspects and completeness of the translation were only secondary goals during development and further improvements are considered. As there is currently only a mapping to SDL defined, additional mapping implementations to other languages are considered. It has been shown that a mapping by means of XSLT is feasible. However, it is not the optimal way for transformations of complex language constructs. Therefore, it might be more practical to consider using Java-Document Object Model (DOM) technology as a replacement in order to process an XMI data structure in the future. However, the downside of this approach is that UML design specifications actually have to be mapped to a target language for analysis, simulation and execution. This raises the desire for an integrated UML development environment that is capable of automatic transformations of UML specifications into SDL.

References

- [AAL+99] D. Amyot, R. Andrade, L. Logrippo, J. Sincennes, Y. Zhimey: Formal Methods for Mobility Standards, in: *Wireless Communications and Systems, 1999 Emerging Technologies Symposium*, Richardson, ISBN 0-7803-5554-7, pp. 14.1-14.7, 1999
- [Abs93] L. Absillis: *Towards Intelligent Computer Network Analysis, A Methodology and its Implementation*, Ph.D. Thesis, Vrije Universiteit Brussel, 1993
- [ACL+04] L. Apvrille, J.-P. Courtiat, C. Lohr, J.-P. de Saqui-Sannes: TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit, *IEEE Transactions on Software Engineering*, Vol. 30, No. 7, pp. 473-487, July 2004
- [Art01] R. Arthaud: *SDL and Layered Systems: Proposed Extensions to SDL to Better Support the Design of Layered Systems*, in: R. Reed, J. Reed (Eds.): *SDL 2001, Lecture Notes in Computer Science (LNCS) Volume 2078*, pp. 52–71, Springer-Verlag Berlin Heidelberg, 2001
- [BBK02] A. Bradley, A. Bestavros, A. Kfoury: *Safe Composition of Web Communication Protocols for Extensible Edge Services*. In: *Proceedings of Workshop on Web Content Caching and Distribution (WCW)*, Boulder, Colorado, 2002
- [BK03] M. Björkander, C. Kobryn: *Architecting System with UML 2.0*, *IEEE Software*, August 2003
http://www.uml-forum.com/docs/papers/IEEE_SW_Jul03_p57_Kobryn.pdf
- [BKV02] S. Bourduas, F. Khendek, D. Vincent: *From MSC and UML to SDL*, *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC '02)*, 0730-3157/2, IEEE Computer Society, 2002
- [BJ78] D. Björner, C. B. Jones (eds.): *The Vienna Development Method: The Meta-Language*, *Lecture Notes in Computer Science*, Vol. 61, Springer-Verlag, 1978
- [BJ98] B. Selic, J. Rumbaugh: *Using UML for Modeling Complex Real-Time Systems*, ObjecTime Ltd., 1998
- [BJ00] B. Selic, J. Raumbaugh: *Mapping SDL to UML*, Rational Software Corp., White Paper, 2000
- [Bjo00] M. Björkander: *Graphical Programming Using UML and SDL*, *IEEE Computer*, Vol. 33, no. 12, pp. 30-35, December 2000
- [BM98] J. Bezivin, P.-A. Muller: *UML: The Birth and Rise of a Standard Modeling Notation*, in: *Selected papers from the First International Workshop on The Unified Modeling Notation UML 98: Beyond the Notation*, pp. 1-8, ISBN 3-540-66252-9, Springer, 1998
- [Boc03a] C. Bock: *UML 2 Activity and Action Models*, *Journal of Object Technology*, Vol. 2, No. 4, pp. 43-53, August 2003

- [Boc03b] C. Bock: UML 2 Activity and Action Models Part 2: Actions, Journal of Object Technology, Vol. 2, No. 5, pp. 41-56, October 2003
- [Boc03c] C. Bock: UML 2 Activity and Action Models Part 3: Control Nodes, Journal of Object Technology, Vol. 2, No. 6, pp. 7-23, December 2003
- [Boc04a] C. Bock: UML 2 Activity and Action Models Part 4: Object Nodes, Journal of Object Technology, Vol. 3, No. 1, pp. 27-41, February 2004
- [Boc04b] C. Bock: UML 2 Activity and Action Models Part 5: Partitions, Journal of Object Technology, Vol. 3, No. 7, pp. 37-56, August 2004
- [Boc05] C. Bock: UML 2 Activity and Action Models Part 6: Structured Activities, Journal of Object Technology, Vol. 4, No. 4, pp. 43-66, June 2005
- [BOW00] M. Bjorkander, I. Ober, T. Weigert: SDL Mapping for the UML Action Semantics, Object Management Group (OMG) document ad/00-08-01, August 2000
<http://www.omg.org/docs/ad/00-08-01.pdf>
- [Bra99] R. Braek et al.: TIME – The integrated method, Sintef Report, 1999
<http://www.sintef.no/time/>
- [BGO+04] M. Bozga, S. Graf, I. Ober, I. Ober, J. Sifakis. Tools and Applications II: The IF Toolset, in: F. Corradinni, M. Bernardo (eds.), Proceedings of SFM'04 (Bertinoro, Italy), Lecture Notes in Computer Science (LNCS) Volume 3185, Springer-Verlag, 2004
- [BGM+01] M. Bozga, S. Graf, L. Mounier, I. Ober, J.-L. Roux, D. Vincent: Timed Extensions for SDL, in: R. Reed, J. Reed (Eds.): SDL 2001, Lecture Notes in Computer Science (LNCS) Volume 2078, pp. 223–241, Springer-Verlag Berlin Heidelberg, 2001
- [BPS+06] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan (eds.): Extensible Markup Language (XML) 1.1 (Second Edition), World Wide Web Consortium (W3C) Recommendation, 16 August 2006
<http://www.w3.org/TR/2006/REC-xml11-20060816/>
- [BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin: Resource ReSerVation Protocol (RSVP) – Version 1 functional specification, Request for Comments 2005, Internet Engineering Task Force (IETF), 1997
- [CB03] K. Y. Chan, G. v. Bochmann: Modeling IETF Session Initiation Protocol and its services in SDL, in: R. Reed (ed.): SDL 2003, Lecture Notes in Computer Science, Vol. 2708, pp. 352-373, Springer-Verlag, Berlin-Heidelberg, 2003
- [CD99] J. Clark, S. DeRose (eds.): XML Path Language (XPath), Version 1, World Wide Web Consortium (W3C) Recommendation, November 1999
<http://www.w3.org/TR/1999/REC-xpath-19991116>

-
- [CGM+04] A. Cavalli, C. Grepet, S. Maag, V. Tortajada: A Validation Model for the DSR Protocol, ICDCS 2004, 2004
- [Dan06] C. Danilov: Performance and functionality in overlay networks, ISBN 04961-6329-9, ProQuest Company, 2006
- [DBS95] J. Davies, J. W. Bryans, S. A. Schneider: Real-time LOTOS and timed observations, in: Formal Description Techniques VIII, Chapman & Hall, 1995
- [Dol01] L. Doldi: SDL Illustrated, Visually design executable models, ISBN 2-9516600-0-6, Paragraphic, 2001
- [Dol03a] L. Doldi: UML 2 Illustrated, Developing Real-Time & Communications Systems, ISBN 2-9516600-1-4, TMSO, 2003
- [Dol03b] L. Doldi: Validation of Communications Systems With Sdl (The Art of Sdl Simulation and Reachability Analysis), ISBN 0470852860, Wiley & Sons, 2003
- [EHS97] J. Ellsberger, D. Hogrefe, A. Sarma: SDL, Formal Object-oriented Language for Communicating Systems, ISBN 0-13-621384-7, Prentice Hall Europe, 1997
- [EM85] H. Ehrig, B. Mahr: Fundamentals of Algebraic Specification, Part 1, Springer Verlag Berlin, 1985
- [EMS00] H. Eirund, B. Müller, G. Schreiber: Formale Beschreibungsverfahren der Informatik, ISBN 3-519-02643-0, Teubner Verlag, 2000
- [ETS05] European Telecommunications Standards Institute: UML Profile for Communicating Systems, ETSI Specification, June 2005
- [FGD+06] R. B. France, S. Gosh, T. Dinh-Trong, A. Solberg: Model-driven development using UML 2.0: promises and pitfalls, in: IEEE Computer Volume 39, Issue 2, pp. 59-66, February 2006
- [FH05] X. Fu, D. Hogrefe: Modeling Soft State Protocols with SDL, in: Proceedings of IFIP International Conference on Networking, Waterloo, Canada, 2005
- [FHL+00] J. Fischer, E. Holz, M. v. Lowis, A. Prinz: SDL-2000: A Language with a Formal Semantics. The Third Workshop on Rigorous Object-Oriented Methods, University of York, UK, 2000
- [GG03] R. Grammes, R. Gotzhein: Towards the Harmonization of UML and SDL, Syntactic and Semantic Alignment, Technical Report 327/03, Computer Science Department, Technical University of Kaiserslautern, Germany, 2003
- [GGP03] U. Glässer, R. Gotzhein, A. Prinz: The formal semantics of SDL-2000: Status and perspectives, in: Computer Networks 42 (2003), pp. 343-358, Elsevier, 2004
- [Got92] R. Gotzhein: Temporal logic and applications – a tutorial. Computer Networks and ISDN Systems, 24, pp. 203-218, 1992

- [Got93] R. Gotzhein: Open Distributed Systems: On Concepts, Methods, and Design from a Logical Point of View. Vieweg-Verlag, Germany, 1993
- [Gra03] S. Graf: Expression of Time and Duration Constraints in SDL, in: E. Sherratt (Ed.): SAM 2002, Lecture Notes in Computer Science (LNCS) Volume 2599, pp. 38–52, Springer-Verlag Berlin Heidelberg, 2003
- [GRS01] J. Grabowski, E. Rudolph, M. Schmitt: Die Spezifikationssprachen MSC und SDL – Teil 1: Message Sequence Chart (MSC), in: at – Automatisierungstechnik 49 (2001) 12, pp. A19-A22, Oldenburg Verlag, 2001
- [Gur88] Y. Gurevich: Logic and the challenge of computer science, in: E. Börger (ed.), Current Trends in Theoretical Computer Science, pp. 1-57, CS Press, 1988
- [Ham05] U. Hammerschall: Verteilte Systeme und Anwendungen, ISBN 3-8273-7096-5, Pearson Studium, 2005
- [Hin98] U. Hinkel: Formale semantische Fundierung und eine darauf abgestützte Verifikationsmethode für SDL, PhD. Thesis, Technische Universität München, 1998
- [HKL+05] R. Hancock, G. Karagiannis, J. Loughney, S. v. d. Bosch: Next Steps in Signaling (NSIS): Framework, Request for Comments 4080, Internet Engineering Task Force (IETF), 2005
- [Hoa04] C. A. R. Hoare: Communicating Sequential Processes, Online Edition, 2004
<http://www.usingcsp.com/cspbook.pdf>
- [Hol91] G. J. Holzmann: Design and Validation of Computer Protocols, ISBN 0-13-539834-7, Prentice Hall, 1991
<http://spinroot.com/spin/Doc/Book91.html>
- [Hog89] D. Hogrefe: Estelle, LOTOS und SDL Standard - Spezifikationssprachen für verteilte Systeme. Springer-Verlag, Berlin, 1989
- [Hog91] D. Hogrefe: OSI Formal Specification Case Study: The Inres Protocol and Service (revised). Technical Report IAM-91-012, Universität Bern, Institut für Informatik, May 1991
- [HR00] D. Harel, B. Rumpe: Modeling Languages: Syntax, Semantics and All That Stuff (Part I: The Basic Stuff). Technical Report MCS00-16, Mathematics & Computer Science, Weizmann Institute of Science, 2000
- [HMU02] J. E. Hopcroft, R. Motwani, J. D. Ullman: Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, 2nd revised edition, ISBN 3-8273-7020-5, Pearson Studium, 2002
- [Koe03] H. König: Protocol Engineering – Prinzip, Beschreibung und Entwicklung von Kommunikationsprotokollen, 1. Auflage, ISBN 3-519-00454-2, Teubner Verlag, November 2003

- [Kay06] M. Kay: SAXON – The XSLT and XQuery Processor, <http://saxon.sourceforge.net/>
- [KAL+01] H. Kaaranen et al.: UMTS Networks – Architecture, Mobility and Services, ISBN 0-4714-8654-X, Wiley & Sons, England, 2001
- [KK02] D. Karagiannis, H. Kühn: Metamodelling Platforms, in: K. Bauknecht, A. Min Tjoa, G. Quirchmayer (Eds.): Proceedings of the Third International Conference EC-Web 2002 – Dexa 2002, Aix-en-Provence, France, Lecture Notes in Computer Science (LNCS) Volume 2455, p.182, Springer-Verlag, 2002
- [KR04] J. Kurose, K. Ross: Computer Networking: A Top-Down Approach Featuring the Internet, 3rd edition, ISBN 0-3212-2735-2, Addison-Wesley, 2004
- [KLP+04] V. D. Kollias, Q. Li, A. Prinz, W. Skelton, A. Yiannakoulias, and K. Moss: Back to the Basics, Poster Presentation at Fourth SDL and MSC Workshop (SAM 2004), 2004
<http://www.site.uottawa.ca/sam04/pres/Prinz.pdf>
- [KPK+03] C. Kavadias, B. Perrin, V. Kollias, M. Loupis: Enhanced SDL Subset for the Design and Implementation of Java-Enabled Embedded Signalling Systems, in: R. Reed, J. Reed (Eds.): SDL 2003, Lecture Notes in Computer Science (LNCS) Volume 2708, pp. 137–149, Springer-Verlag Berlin Heidelberg, 2003
- [Lar99] J. Larmouth: ASN.1 Complete, ISBN 0-12233-435-3, Morgan Kaufmann Publishers, Elsevier, 1999
<http://www.oss.com/asn1/larmouth.html>
- [Lie03] G. Lienemann: TCP/IP – Grundlagen, Protokolle und Routing, ISBN 3-936931-07-0, Heise Zeitschriften Verlag, Hannover, 2003
- [Loe03] M. von Löwis of Menar: Formale Semantik des Datentypmodells von SDL-2000, Ph.D. Thesis, Humboldt-Universität Berlin, 2003
- [LW93] B. Liskov, J. M. Wing: Family Values: A Behavioral Notion of Subtyping. Technical Report CMU-CS-93-187, Computer Science Department, Carnegie Mellon University, Pittsburg, 1993
- [ISO84] International Standards Organization: Information Processing Systems – Open Systems Interconnection – Basic Reference Model, ISO 7494, 1984
- [ISO89] International Standards Organization: Information processing systems - Open Systems Interconnection – LOTOS - A formal description technique based on the temporal ordering of observational behaviour, International Standards Organization, ISO/IEC 8807(E), 1989
- [ISO97] International Standards Organization: Estelle - a formal description technique based on an extended state transition model, International Standards Organization, ISO/IEC 9074(E), 1997
- [ITU02a] International Telecommunication Union: Specification and Description Language (SDL), ITU-T Recommendation Z.100, revised, August 2002

- [ITU02b] International Telecommunication Union: Common Interchange Format for SDL, ITU-T Recommendation Z.106, revised, August 2002
- [ITU03] International Telecommunication Union: Testing and Test Control Notation version 3 (TTCN-3): Core language, ITU-T Recommendation Z.140, April 2003
- [ITU06a] International Telecommunication Union: UML Profile for SDL (Input for Z.109 revision) TDT09r17, Temporary Document TDX17, September 2006
- [ITU06b] International Telecommunication Union: Notations to Define ITU-T languages, Input for Z.111 draft, Temporary Document TDX 21, June 2006
- [ITU01] International Telecommunication Union: Message Sequence Charts (MSC), ITU-T Recommendation Z.120, October 2001
- [ITU99] International Telecommunication Union: SDL combined with UML, ITU-T Recommendation Z.109, November 1999
- [JGK+03] P. Ji, Z. Ge, J. Kurose, D. Towsly: A comparison of hard-state and soft-state signaling protocols, in: Proceedings of SIGCOMM 2003, Karlsruhe, Germany, 2003
- [Pri01] A. Prinz: Formal Semantics for SDL, Definition and Implementation, Habilitation Thesis, Humboldt-University Berlin, May 2001
- [Mag] MagicDraw 11.0, NoMagic Inc., <http://www.magicdraw.com/>
- [Mal94] G. Malkin: RIP Version 2 – Carrying Additional Information, Request for Comments 1723, Internet Engineering Task Force (IETF), 1994
- [MB02] S. J. Mellor, M. J. Balcer: Executable UML: A Foundation for Model-Driven Architecture, Addison Wesley, ISBN 0-2017-4804-5, 2002
- [Mer01] S. Merz: Model Checking: A Tutorial Overview, in: F. Cassez et al. (eds): Modeling and Verification of Parallel Processes, Springer-Verlag, Lecture Notes in Computer Science (LNCS) Volume 2067, pp. 3-38, 2001
- [Mil80] R. Milner: A calculus of communicating systems, Springer-Verlag, ISBN 0-3871-0235-5, 1980
- [Mol00] B. Moller-Pedersen: SDL Combined with UML, in: Telektronikk, Volume 4, Languages for Telecommunications Applications, ISSN 0085-7130, 2000
- [MM04] C. S. R. Muthy, B. S. Manoj: Ad Hoc Wireless Networks: Architectures and Protocols, ISBN 0-1314-7023-X, Prentice Hall, 2004
- [MSP01] O. Monkewich, I. Sales, R. Probert: OSPF Efficient LSA Refreshment Function in SDL, in: Reed, R., Reed, J. (eds.): SDL 2001, Lecture Notes in Computer Science, Vol. 2078, pp. 300–315, Springer-Verlag, Berlin Heidelberg New York, 2001

-
- [Obe01] I. Ober: Specification and Validation of Timed Systems using Formal Description Languages. PhD. Thesis, Institute National Polytechnique de Toulouse, France, 2001
- [OMG03a] Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification, ptc/04-10-15, October 2003
<http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf>
- [OMG03b] Object Management Group: MDA Guide Version 1.0.1, omg/03-06-01, June 2003
<http://www.omg.org/docs/omg/03-06-01.pdf>
- [OMG04a] Object Management Group: Unified Modeling Language: Infrastructure version 2.0, ptc/04-10-14, November 2004
<http://www.omg.org/cgi-bin/doc?ptc/04-10-14>
- [OMG04b] Object Management Group: Human-Usable Textual Notation (HUTN) Specification, V1.0, formal/04-08-01, August 2004
<http://www.omg.org/cgi-bin/doc?formal/2004-08-01>
- [OMG05a] Object Management Group: Unified Modeling Language: Superstructure version 2.0, formal/05-07-04, August 2005
<http://www.omg.org/cgi-bin/doc?formal/05-07-04>
- [OMG05b] Object Management Group: UML Testing Profile Version 1.0, formal/05-07-07, July 2005
<http://www.omg.org/cgi-bin/doc?formal/05-07-07>
- [OMG05c] Object Management Group: UML Profile for Schedulability, Performance, and Time Specification Version 1.1, formal/05-01-02, January 2005
<http://www.omg.org/cgi-bin/doc?formal/2005-01-02>
- [OMG05d] Object Management Group: OCL 2.0 Specification, Version 2.0, ptc/2005-06-06, June 2005
<http://www.omg.org/cgi-bin/doc?ptc/2005-06-06>
- [OMG05e] Object Management Group: Meta Object Facility (MOF) 2.0 XMI Mapping Specification, v2.1, formal/05-09-01, September 2005
<http://www.omg.org/cgi-bin/apps/doc?formal/05-09-01.pdf>
- [OMG06] Object Management Group: Unified Modeling Language: Superstructure version 2.1, ptc/06-04-02, April 2006
- [PD03] L. Peterson, B. Davie: Computernetzwerke – Eine systemorientierte Einführung, Deutsche Ausgabe der 3. Auflage, ISBN 3-89864-242-9, 2003
- [Pri03] A. Prinz: SDL Time Extensions from a Semantic Point of View, in: E. Sherratt (Ed.): SAM 2002, Lecture Notes in Computer Science (LNCS) Volume 2599, pp. 38–52, Springer-Verlag Berlin Heidelberg, 2003
- [Rec04] J. Rech: Wireless LANs, 802.11-WLAN-Technologie und praktische Umsetzung im Detail. ISBN 3-936931-04-6, Heise Zeitschriften Verlag Hannover, 2004

- [Ree00] R. Reed: SDL-2000 for New Millennium Systems, in: *Teletronikk 2000* (4), pp. 20-35, 2000
- [RHQ+05] C. Rupp, J. Hahn, S. Queins, M. Jeckle, B. Zengler: *UML2 glasklar*, 2nd edition, ISBN 3-446-22952-3, Hanser Verlag Munich Vienna, 2005
- [RM99] S. Raman, S. McCanne: A model, analysis and protocol framework for soft state-based communication, in: *Proceedings of SIGCOMM 1999*, Cambridge, MA, 1999
- [RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo et al.: SIP: Session Initiation Protocol, Request for Comments 3261, Internet Engineering Task Force (IETF), June 2002
- [SEF+97] P. Sharma, D. Estrin, S. Floyd, V. Jacobson: Scalable timers for soft state protocols, in: *InfoCom 97*, Kobe, Japan, 1997
- [Sch03a] R. Schröder: *SDL Datenkonzepte – Analyse und Verbesserungen*, Ph.D. Thesis, Humboldt University of Berlin, Germany, 2003
- [Sch03b] J. Schiller: *Mobile Communications*, 2nd edition, ISBN 03211-2381-6, Addison-Wesley, 2003
- [SCF+03] H. Schulzrinne, S. Casner, B. Frederik, V. Jacobson: RTP: A Transport Protocol for Real-Time Systems, Request for Comments 3550, Internet Engineering Task Force (IETF), July 2003
- [Sel04] B. Selic: On the Semantic Foundations of Standard UML 2.0, In: M. Bernardo, F. Corradini (Eds.): *SFM-RT 2004*, Lecture Notes in Computer Science (LNCS) Volume 3185, pp. 181–199, 2004
- [Sel98] B. Selic: Using UML for Modeling Complex Real-Time Systems, In: F. Mueller, A. Bestavros (Eds.), *Proceedings of ACM SIGPLAN Workshop LCTES'98*, Montreal, Canada, Lecture Notes in Computer Science (LNCS) Volume 1474/1998, pp. 250, June 1998
- [She05] E. Sherrat: *SDL in a Changing World*, In: D. Amyot and A.W. Williams (Eds.): *SAM 2004*, Lecture Notes in Computer Science (LNCS) Volume 3319, pp. 96–105, Springer-Verlag Berlin Heidelberg, 2005
- [SGW94] B. Selic, G. Gullekson, P. Ward: *Real-Time Object-Oriented Modeling*. John Wiley & Sons, New York, NY, 1994
- [SK95] K. Slonneger, B. L. Kurtz: *Formal syntax and semantics of programming languages: a laboratory based approach*. ISBN 0-201-65697-3, Addison-Wesley Publishing Company, 1995
<http://www.cs.uiowa.edu/~slonnegr/plf/Book/>

-
- [SWH06] R. Soltwisch, C. Werner, D. Hogrefe: A new Formal Methodology for Multi-Role Nodes: Specification and Validation of an IDKE aware Access Router, to appear in: The Proceedings of the IEEE International Conference on Networks (ICON) 2006, Singapore, October 2006
- [Tan02] A. S. Tanenbaum: Computer Networks, Forth Edition, ISBN 0-13-066102-3, Prentice Hall, 2002
- [Tau] Telelogic Tau SDL Suite 4.6, Telelogic A.B., <http://www.telelogic.com>
- [TS03] A. Tanenbaum, M. v. Steen: Verteilte System, Grundlagen und Pradigmen, ISBN 3-8273-7054-4, Pearson Studium, 2003
- [VE99] K. Verschaeve, A. Ek: Three Scenarios for Combining UML and SDL 96, in: Proceedings of SDL Forum '99, Montréal, Canada, June 1999
- [Ver01a] K. Verschaeve: Combining UML and SDL, Proceedings of Workshop on Transformations in the Unified Modeling Language (WTUML01), ETAPS 2001, Genova, Italy, 2001
- [Ver01b] K. Verschaeve: UML - SDL Round-trip Engineering through Incremental Translation of Changes. PhD. Thesis, Vrije University of Brussel, 2001
- [Wet04] N. de Wet: Model Driven Communication Engineering and Simulation based Performance Analysis using UML 2.0, PhD. Thesis, University of Cape Town, South Africa, 2004
- [WFH05] C. Werner, X. Fu, D. Hogrefe: Modeling Route Change in Soft State Signaling Protocols Using SDL: a Case of RSVP, in: A. Prinz, R. Reed and J. Reed (eds.), Proceedings of the 12th SDL Forum (SDL 2005), Grimstad, Norway, Lecture Notes in Computer Science (LNCS) Volume 3530, pp. 174-186, Springer Verlag, ISBN 3-540-26612-7, June 2005
- [WH06] C. Werner, D. Hogrefe: UML Profile for Communicating Systems, Technical Report No. IFI-TB-2006-03, Institute for Informatics, University of Göttingen, Germany, ISSN 1611-1044, March 2006
- [WKH06] C. Werner, S. Kraatz, D. Hogrefe: UML Profile for Communicating Systems, in: Proceedings of the Fifth Workshop on System Analysis and Modelling (SAM 06), Kaiserslautern, Germany, pp. 81-90, June 2006
- [Xal] The Apache XML Project: Xalan, <http://xalan.apache.org/index.html>
- [ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala: RSVP: a New Resource Reservation Protocol, IEEE Network, 1993

Abbreviations

ADT	Abstract Data Type
AS0	Abstract Syntax 0
AS1	Abstract Syntax 1
ASE	Application Service Elements
ASM	Abstract State Machine
ASN.1	Abstract Syntax Notation number One
BNF	Backus-Naur Form
DOM	Document Object Model
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CEFSM	Communicating Extended Finite State Machine
CIF	Common Interchange Format
CSP	Communicating Sequential Processes
EBNF	Extended Backus-Naur Form
EFSM	Extended Finite State Machine
ETSI	European Telecommunications Standards Institute
FDT	Formal Description Techniques
FIFO	First-in first-out
FSM	Finite State Machine
HUTN	Human-usable Textual Notation
ICI	Interface Control Information
IDU	Interface Data Unit
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
J2EE	Java Platform, Enterprise Edition
LHS	Left-hand-side rules
LSP	Liskov Substitution Principle
MANET	Mobile Ad-hoc Networks
MDA	Model Driven Architecture
MDD	Model Driven Development
MOF	Meta-Object Facility

MSC	Message Sequence Charts
NSIS	Next Steps in Signaling
OCL	Object Constraint Language
OMG	Object Management Group
OMT	Object Modeling Technique
OOA&D	Object Oriented Analysis and Design
OOD	Object Oriented Design
OOSE	Object-Oriented Software Engineering
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
Pid	Process Identification
PIM	Platform Independent Model
PSM	Platform Specific Model
RFC	Request for Comments
RHS	Right-hand-side rules
RIP	Routing Information Protocol
ROOM	Real-time Object-oriented Modeling
RPC	Remote Procedure Call
RSVP	Resource Reservation Protocol
RTC	Run-to-Completion
RTCP	RealTime Control Protocol
SAP	Service Access Point
SDL	Specification and Description Language
SDL/GR	SDL Graphic Representation
SDL/PR	SDL Phrase Representation
SDU	Service Data Unit
SIP	Session Initiation Protocol
SNA	Systems Network Architecture
TCP	Transmission Control Protocol
TTCN	Testing and Test Control Notation
UDP	User Datagram Protocol
UML	Unified Modeling Language
UML CS	UML Profile for Communicating Systems
UMTS	Universal Mobile Telecommunication System

W3C	World Wide Web Consortium
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless LAN
VDM	Vienna Development Method
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Appendix A: UML CS Profile in XMI 2.1

The following lists the UML CS profile exported from MagicDraw 11.0 [Mag] to XMI 2.1 format. It can be imported to this tool and applied to new UML modeling projects. For other UML 2 and XMI 2.1 compliant modeling tools, it should also be possible to import this profile. However, at the time of writing, there is no UML modeling tool known capable of importing XMI 2.1-based profiles.

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- <!DOCTYPE XMI SYSTEM "uml14xmi12.dtd" --> -->

<xmi:XMI xmi:version='2.1' timestamp='Thu Aug 10 12:33:47 CEST 2006'
  xmlns:uml='http://schema.omg.org/spec/UML/2.0'
  xmlns:xmi='http://schema.omg.org/spec/XMI/2.1'>
  <xmi:Documentation xmi:Exporter='MagicDraw UML' xmi:ExporterVersion='11.0' />
  <uml:Model xmi:id='eee_1045467100313_135436_1' name='Data' visibility='public'>
    <ownedMember xmi:type='uml:Profile' xmi:id='_10_5_1_df009b_1140096594546_163685_2'
      name='UML CS Profile' visibility='public' nestingPackage='eee_1045467100313_135436_1'
      owningPackage='eee_1045467100313_135436_1'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
        <moduleExtension moduleRoot='::UML CS Profile' />
      </xmi:Extension>
      <ownedStereotype xmi:type='uml:Stereotype' xmi:id='_10_5_1_df009b_1140096615312_13469_3'
        name='state' visibility='public'>
        <nestedClassifier xmi:type='uml:Extension'
          xmi:id='_10_5_1_df009b_1140097157265_84473_75' visibility='public'
          UMLClass='_10_5_1_df009b_1140096615312_13469_3'>
          <memberEnd xmi:idref='_10_5_1_df009b_1140097157265_914544_76' />
          <memberEnd xmi:idref='_10_5_1_df009b_1140097157265_740458_77' />
          <ownedEnd xmi:type='uml:ExtensionEnd'
            xmi:id='_10_5_1_df009b_1140097157265_740458_77' name='extension$state'
            visibility='private' owningAssociation='_10_5_1_df009b_1140097157265_84473_75'
            association='_10_5_1_df009b_1140097157265_84473_75'
            type='_10_5_1_df009b_1140096615312_13469_3'>
            <upperValue xmi:type='uml:LiteralInteger'
              xmi:id='_10_5_1_df009b_1140097157265_377995_79' value='1' visibility='public'
              owningUpper='_10_5_1_df009b_1140097157265_740458_77' />
            <lowerValue xmi:type='uml:LiteralInteger'
              xmi:id='_10_5_1_df009b_1140097157265_752958_78' visibility='public'
              owningLower='_10_5_1_df009b_1140097157265_740458_77' />
            </ownedEnd>
          </nestedClassifier>
          <ownedAttribute xmi:type='uml:Property'
            xmi:id='_10_5_1_df009b_1140097157265_914544_76' name='base$State' visibility='private'
            UMLClass='_10_5_1_df009b_1140096615312_13469_3'
            association='_10_5_1_df009b_1140097157265_84473_75'>
            <type xmi:type='uml:Class'
              href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932587_977586_9575'>
              <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
                11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
                  Metamodel::StateMachines::BehaviorStateMachines::State' referentType='Class' />
              </xmi:Extension>
            </type>
            </ownedAttribute>
            <ownedAttribute xmi:type='uml:Property'
              xmi:id='_11_0_1_df009b_1147179236673_844291_218' name='isStateList'
              visibility='private' UMLClass='_10_5_1_df009b_1140096615312_13469_3'>
              <type xmi:type='uml:DataType'
                href='UML_Standard_Profile.xml|eee_1045467100323_191782_59'>
                <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
                  11.0'>
                  <referenceExtension referentPath='UML Standard Profile::UML Standard
                    Profile::datatypes::boolean' referentType='DataType' />
                </xmi:Extension>
              </type>
              <defaultValue xmi:type='uml:LiteralBoolean'
                xmi:id='_11_0_1_df009b_1147179249001_783674_219' visibility='public'
                owningProperty='_11_0_1_df009b_1147179236673_844291_218' />
              </ownedAttribute>
```

```

</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140096649906_542934_9' name='start' visibility='public'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
    <modelExtension>
      <_constraintOfConstrainedElement
        xmi:idref='_10_5_1_df009b_1140097106609_459590_74'>
      </modelExtension>
    </xmi:Extension>
    <nestedClassifier xmi:type='uml:Extension'
      xmi:id='_10_5_1_df009b_1140097084640_309636_69' visibility='public'
      UMLClass='_10_5_1_df009b_1140096649906_542934_9'>
      <memberEnd xmi:idref='_10_5_1_df009b_1140097084640_960382_70'>
      <memberEnd xmi:idref='_10_5_1_df009b_1140097084640_441099_71'>
      <ownedEnd xmi:type='uml:ExtensionEnd'
        xmi:id='_10_5_1_df009b_1140097084640_441099_71' name='extension$start'
        visibility='private' owningAssociation='_10_5_1_df009b_1140097084640_309636_69'
        association='_10_5_1_df009b_1140097084640_309636_69'
        type='_10_5_1_df009b_1140096649906_542934_9'>
        <upperValue xmi:type='uml:LiteralInteger'
          xmi:id='_10_5_1_df009b_1140097084656_316583_73' value='1' visibility='public'
          owningUpper='_10_5_1_df009b_1140097084640_441099_71'>
        <lowerValue xmi:type='uml:LiteralInteger'
          xmi:id='_10_5_1_df009b_1140097084640_495713_72' visibility='public'
          owningLower='_10_5_1_df009b_1140097084640_441099_71'>
        </ownedEnd>
      </nestedClassifier>
      <ownedAttribute xmi:type='uml:Property'
        xmi:id='_10_5_1_df009b_1140097084640_960382_70' name='base$Pseudostate'
        visibility='private' UMLClass='_10_5_1_df009b_1140096649906_542934_9'
        association='_10_5_1_df009b_1140097084640_309636_69'>
        <type xmi:type='uml:Class'
          href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
          <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
            11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
              Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'>
            </xmi:Extension>
          </type>
        </ownedAttribute>
        <ownedRule xmi:type='uml:Constraint' xmi:id='_10_5_1_df009b_1140097106609_459590_74'
          name='unnamed1' visibility='public'
          constrainedElement='_10_5_1_df009b_1140096649906_542934_9'
          namespace='_10_5_1_df009b_1140096649906_542934_9'>
        </ownedRule>
      </ownedStereotype>
      <ownedStereotype xmi:type='uml:Stereotype'
        xmi:id='_10_5_1_df009b_1140096667875_193707_15' name='stop' visibility='public'>
        <nestedClassifier xmi:type='uml:Extension'
          xmi:id='_10_5_1_df009b_1140102679250_479321_380' visibility='public'
          UMLClass='_10_5_1_df009b_1140096667875_193707_15'>
          <memberEnd xmi:idref='_10_5_1_df009b_1140102679250_21455_381'>
          <memberEnd xmi:idref='_10_5_1_df009b_1140102679250_132869_382'>
          <ownedEnd xmi:type='uml:ExtensionEnd'
            xmi:id='_10_5_1_df009b_1140102679250_132869_382' name='extension$stop'
            visibility='private' owningAssociation='_10_5_1_df009b_1140102679250_479321_380'
            association='_10_5_1_df009b_1140102679250_479321_380'
            type='_10_5_1_df009b_1140096667875_193707_15'>
            <upperValue xmi:type='uml:LiteralInteger'
              xmi:id='_10_5_1_df009b_1140102679250_857072_384' value='1' visibility='public'
              owningUpper='_10_5_1_df009b_1140102679250_132869_382'>
            <lowerValue xmi:type='uml:LiteralInteger'
              xmi:id='_10_5_1_df009b_1140102679250_467553_383' visibility='public'
              owningLower='_10_5_1_df009b_1140102679250_132869_382'>
            </ownedEnd>
          </nestedClassifier>
          <ownedAttribute xmi:type='uml:Property'
            xmi:id='_10_5_1_df009b_1140102679250_21455_381' name='base$Pseudostate'
            visibility='private' UMLClass='_10_5_1_df009b_1140096667875_193707_15'
            association='_10_5_1_df009b_1140102679250_479321_380'>
            <type xmi:type='uml:Class'
              href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
              <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
                11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
                  Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'>
                </xmi:Extension>
              </type>
            </ownedAttribute>
          </nestedClassifier>
        </ownedStereotype>
      </xmi:Extension>
    </xmi:Extension>
  </xmi:Extension>

```

```

    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140096710203_818591_21' name='stateMachine'
  visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
  xmi:id='_10_5_1_df009b_1140097180937_207176_80' visibility='public'
  UMLClass='_10_5_1_df009b_1140096710203_818591_21'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140097180937_527500_81'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140097180937_936262_82'>
    <ownedEnd xmi:type='uml:ExtensionEnd'
  xmi:id='_10_5_1_df009b_1140097180937_936262_82' name='extension$stateMachine'
  visibility='private' owningAssociation='_10_5_1_df009b_1140097180937_207176_80'
  association='_10_5_1_df009b_1140097180937_207176_80'
  type='_10_5_1_df009b_1140096710203_818591_21'>
    <upperValue xmi:type='uml:LiteralInteger'
  xmi:id='_10_5_1_df009b_1140097180937_631903_84' value='1' visibility='public'
  owningUpper='_10_5_1_df009b_1140097180937_936262_82'>
    <lowerValue xmi:type='uml:LiteralInteger'
  xmi:id='_10_5_1_df009b_1140097180937_119906_83' visibility='public'
  owningLower='_10_5_1_df009b_1140097180937_936262_82'>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
  xmi:id='_10_5_1_df009b_1140097180937_527500_81' name='base$StateMachine'
  visibility='private' UMLClass='_10_5_1_df009b_1140096710203_818591_21'
  association='_10_5_1_df009b_1140097180937_207176_80'>
    <type xmi:type='uml:Class'
  href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932656_223024_9583'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
  11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
  Metamodel::StateMachines::BehaviorStateMachines::StateMachine' referentType='Class'>
    </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140096721328_485203_27' name='region' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
  xmi:id='_10_5_1_df009b_1140097278984_299776_95' visibility='public'
  UMLClass='_10_5_1_df009b_1140096721328_485203_27'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140097278984_773752_96'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140097278984_479139_97'>
    <ownedEnd xmi:type='uml:ExtensionEnd'
  xmi:id='_10_5_1_df009b_1140097278984_479139_97' name='extension$region'
  visibility='private' owningAssociation='_10_5_1_df009b_1140097278984_299776_95'
  association='_10_5_1_df009b_1140097278984_299776_95'
  type='_10_5_1_df009b_1140096721328_485203_27'>
    <upperValue xmi:type='uml:LiteralInteger'
  xmi:id='_10_5_1_df009b_1140097278984_58017_99' value='1' visibility='public'
  owningUpper='_10_5_1_df009b_1140097278984_479139_97'>
    <lowerValue xmi:type='uml:LiteralInteger'
  xmi:id='_10_5_1_df009b_1140097278984_284243_98' visibility='public'
  owningLower='_10_5_1_df009b_1140097278984_479139_97'>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
  xmi:id='_10_5_1_df009b_1140097278984_773752_96' name='base$Region' visibility='private'
  UMLClass='_10_5_1_df009b_1140096721328_485203_27'
  association='_10_5_1_df009b_1140097278984_299776_95'>
    <type xmi:type='uml:Class'
  href='UML_Standard_Profile.xml|_9_0_62a020a_1105704933219_299257_9640'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
  11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
  Metamodel::StateMachines::BehaviorStateMachines::Region' referentType='Class'>
    </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140096733296_708315_33' name='transition' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
  xmi:id='_10_5_1_df009b_1140097256046_625081_90' visibility='public'
  UMLClass='_10_5_1_df009b_1140096733296_708315_33'>

```

```

    <memberEnd xmi:idref='_10_5_1_df009b_1140097256046_438496_91' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140097256046_954860_92' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140097256046_954860_92' name='extension$transition'
visibility='private' owningAssociation='_10_5_1_df009b_1140097256046_625081_90'
association='_10_5_1_df009b_1140097256046_625081_90'
type='_10_5_1_df009b_1140096733296_708315_33'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097256046_282199_94' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140097256046_954860_92' />
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097256046_798222_93' visibility='public'
owningLower='_10_5_1_df009b_1140097256046_954860_92' />
    </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140097256046_438496_91' name='base$Transition'
visibility='private' UMLClass='_10_5_1_df009b_1140096733296_708315_33'
association='_10_5_1_df009b_1140097256046_625081_90'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932761_361072_9592'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Transition' referentType='Class' />
        </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140096751750_315342_39' name='decision' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140097421000_257453_120' visibility='public'
UMLClass='_10_5_1_df009b_1140096751750_315342_39'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140097421000_981474_121' />
        <memberEnd xmi:idref='_10_5_1_df009b_1140097421000_507574_122' />
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140097421000_507574_122' name='extension$decision'
visibility='private' owningAssociation='_10_5_1_df009b_1140097421000_257453_120'
association='_10_5_1_df009b_1140097421000_257453_120'
type='_10_5_1_df009b_1140096751750_315342_39'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097421000_793459_124' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140097421000_507574_122' />
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097421000_993408_123' visibility='public'
owningLower='_10_5_1_df009b_1140097421000_507574_122' />
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140097421000_981474_121' name='base$Pseudostate'
visibility='private' UMLClass='_10_5_1_df009b_1140096751750_315342_39'
association='_10_5_1_df009b_1140097421000_257453_120'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class' />
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140096771468_833357_45' name='merge' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140097358671_851220_110' visibility='public'
UMLClass='_10_5_1_df009b_1140096771468_833357_45'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140097358671_534035_111' />
        <memberEnd xmi:idref='_10_5_1_df009b_1140097358671_760710_112' />
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140097358671_760710_112' name='extension$merge'
visibility='private' owningAssociation='_10_5_1_df009b_1140097358671_851220_110'
association='_10_5_1_df009b_1140097358671_851220_110'
type='_10_5_1_df009b_1140096771468_833357_45'>
    
```

```

    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097358671_123788_114' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140097358671_760710_112'>/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097358671_729518_113' visibility='public'
owningLower='_10_5_1_df009b_1140097358671_760710_112'>/>
</ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140097358671_534035_111' name='base$Pseudostate'
visibility='private' UMLClass='_10_5_1_df009b_1140096771468_833357_45'
association='_10_5_1_df009b_1140097358671_851220_110'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'>/>
        </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140096853281_52313_51' name='methodStart' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140097307937_455213_100' visibility='public'
UMLClass='_10_5_1_df009b_1140096853281_52313_51'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140097307937_940751_101'>/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140097307937_38939_102'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140097307937_38939_102' name='extension$methodStart'
visibility='private' owningAssociation='_10_5_1_df009b_1140097307937_455213_100'
association='_10_5_1_df009b_1140097307937_455213_100'
type='_10_5_1_df009b_1140096853281_52313_51'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097307937_554709_104' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140097307937_38939_102'>/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097307937_131199_103' visibility='public'
owningLower='_10_5_1_df009b_1140097307937_38939_102'>/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140097307937_940751_101' name='base$Pseudostate'
visibility='private' UMLClass='_10_5_1_df009b_1140096853281_52313_51'
association='_10_5_1_df009b_1140097307937_455213_100'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'>/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140096871062_920988_57' name='methodReturn'
visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140097335328_734478_105' visibility='public'
UMLClass='_10_5_1_df009b_1140096871062_920988_57'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140097335328_180221_106'>/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140097335328_393312_107'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140097335328_393312_107' name='extension$methodReturn'
visibility='private' owningAssociation='_10_5_1_df009b_1140097335328_734478_105'
association='_10_5_1_df009b_1140097335328_734478_105'
type='_10_5_1_df009b_1140096871062_920988_57'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097335328_916362_109' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140097335328_393312_107'>/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097335328_484195_108' visibility='public'
owningLower='_10_5_1_df009b_1140097335328_393312_107'>/>
        </ownedEnd>
    </nestedClassifier>

```

```

    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140097335328_180221_106' name='base$Pseudostate'
visibility='private' UMLClass='_10_5_1_df009b_1140096871062_920988_57'
association='_10_5_1_df009b_1140097335328_734478_105'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'/>
    </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140096902015_225203_63' name='history' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140097382578_268126_115' visibility='public'
UMLClass='_10_5_1_df009b_1140096902015_225203_63'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140097382578_710474_116' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140097382578_231272_117' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140097382578_231272_117' name='extension$history'
visibility='private' owningAssociation='_10_5_1_df009b_1140097382578_268126_115'
association='_10_5_1_df009b_1140097382578_268126_115'
type='_10_5_1_df009b_1140096902015_225203_63'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097382578_162854_119' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140097382578_231272_117' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140097382578_11115_118' visibility='public'
owningLower='_10_5_1_df009b_1140097382578_231272_117' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140097382578_710474_116' name='base$Pseudostate'
visibility='private' UMLClass='_10_5_1_df009b_1140096902015_225203_63'
association='_10_5_1_df009b_1140097382578_268126_115'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140100078453_319819_284' name='compositeState'
visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140100127406_746646_290' visibility='public'
UMLClass='_10_5_1_df009b_1140100078453_319819_284'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140100127406_367739_291' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140100127406_162060_292' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140100127406_162060_292' name='extension$compositeState'
visibility='private' owningAssociation='_10_5_1_df009b_1140100127406_746646_290'
association='_10_5_1_df009b_1140100127406_746646_290'
type='_10_5_1_df009b_1140100078453_319819_284'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140100127406_658823_294' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140100127406_162060_292' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140100127406_991363_293' visibility='public'
owningLower='_10_5_1_df009b_1140100127406_162060_292' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140100127406_367739_291' name='base$State' visibility='private'
UMLClass='_10_5_1_df009b_1140100078453_319819_284'
association='_10_5_1_df009b_1140100127406_746646_290'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932587_977586_9575'>

```



```

    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::State' referentType='Class'/>
    </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140102836562_575475_395' name='signal' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140102864515_97155_401' visibility='public'
UMLClass='_10_5_1_df009b_1140102836562_575475_395'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140102864515_666749_402'/>
    <memberEnd xmi:idref='_10_5_1_df009b_1140102864515_406446_403'/>
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140102864515_406446_403' name='extension$signal'
visibility='private' owningAssociation='_10_5_1_df009b_1140102864515_97155_401'
association='_10_5_1_df009b_1140102864515_97155_401'
type='_10_5_1_df009b_1140102836562_575475_395'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140102864515_659858_405' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140102864515_406446_403'/>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140102864515_85455_404' visibility='public'
owningLower='_10_5_1_df009b_1140102864515_406446_403'/>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140102864515_666749_402' name='base$Signal'
visibility='private' UMLClass='_10_5_1_df009b_1140102836562_575475_395'
association='_10_5_1_df009b_1140102864515_97155_401'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147183307345_625916_220' name='priority' visibility='public'
UMLClass='_10_5_1_df009b_1140102836562_575475_395'
type='_11_0_1_df009b_1147170006673_663635_167'>
    <defaultValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147183322517_341483_221' visibility='public'
owningProperty='_11_0_1_df009b_1147183307345_625916_220'/>
  </ownedAttribute>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147183333751_115485_222' isReadOnly='true' name='sender'
visibility='public' UMLClass='_10_5_1_df009b_1140102836562_575475_395'
type='_11_0_1_df009b_1147169951783_357389_163'/>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140105350421_811237_556' name='entryPoint' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140105465968_517671_562' visibility='public'
UMLClass='_10_5_1_df009b_1140105350421_811237_556'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140105465968_831630_563'/>
    <memberEnd xmi:idref='_10_5_1_df009b_1140105465968_658299_564'/>
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140105465968_658299_564' name='extension$entryPoint'
visibility='private' owningAssociation='_10_5_1_df009b_1140105465968_517671_562'
association='_10_5_1_df009b_1140105465968_517671_562'
type='_10_5_1_df009b_1140105350421_811237_556'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140105465984_91358_566' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140105465968_658299_564'/>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140105465984_417991_565' visibility='public'
owningLower='_10_5_1_df009b_1140105465968_658299_564'/>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140105465968_831630_563' name='base$Pseudostate'

```

```

visibility='private' UMLClass='_10_5_1_df009b_1140105350421_811237_556'
association='_10_5_1_df009b_1140105465968_517671_562'
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140105764687_984929_567' name='exitPoint' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140105811234_498115_573' visibility='public'
UMLClass='_10_5_1_df009b_1140105764687_984929_567'>
  <memberEnd xmi:idref='_10_5_1_df009b_1140105811234_448390_574'/>
  <memberEnd xmi:idref='_10_5_1_df009b_1140105811234_582493_575'/>
  <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140105811234_582493_575' name='extension$exitPoint'
visibility='private' owningAssociation='_10_5_1_df009b_1140105811234_498115_573'
association='_10_5_1_df009b_1140105811234_498115_573'
type='_10_5_1_df009b_1140105764687_984929_567'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140105811234_158967_577' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140105811234_582493_575'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140105811234_511929_576' visibility='public'
owningLower='_10_5_1_df009b_1140105811234_582493_575'/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140105811234_448390_574' name='base$Pseudostate'
visibility='private' UMLClass='_10_5_1_df009b_1140105764687_984929_567'
association='_10_5_1_df009b_1140105811234_498115_573'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704932903_534887_9608'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::StateMachines::BehaviorStateMachines::Pseudostate' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140105846578_762710_578' isActive='true' name='system'
visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140105881125_236603_584' visibility='public'
UMLClass='_10_5_1_df009b_1140105846578_762710_578'>
  <memberEnd xmi:idref='_10_5_1_df009b_1140105881125_953898_585'/>
  <memberEnd xmi:idref='_10_5_1_df009b_1140105881125_807344_586'/>
  <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140105881125_807344_586' name='extension$system'
visibility='private' owningAssociation='_10_5_1_df009b_1140105881125_236603_584'
association='_10_5_1_df009b_1140105881125_236603_584'
type='_10_5_1_df009b_1140105846578_762710_578'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140105881125_538493_588' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140105881125_807344_586'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140105881125_185705_587' visibility='public'
owningLower='_10_5_1_df009b_1140105881125_807344_586'/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140105881125_953898_585' name='base$Class' visibility='private'
UMLClass='_10_5_1_df009b_1140105846578_762710_578'
association='_10_5_1_df009b_1140105881125_236603_584'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885343_144138_7929'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>

```

```

    <referenceExtension referentPath='UML Standard Profile::UML2.0
    Metamodel::Classes::Kernel::Class' referentType='Class'/>
  </xmi:Extension>
</type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140106056406_725870_600' name='block' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140106094390_87790_606' visibility='public'
    UMLClass='_10_5_1_df009b_1140106056406_725870_600'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140106094390_329364_607' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140106094390_990369_608' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_10_5_1_df009b_1140106094390_990369_608' name='extension$block'
      visibility='private' owningAssociation='_10_5_1_df009b_1140106094390_87790_606'
      association='_10_5_1_df009b_1140106094390_87790_606'
      type='_10_5_1_df009b_1140106056406_725870_600'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106094390_914428_610' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1140106094390_990369_608' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106094390_924261_609' visibility='public'
        owningLower='_10_5_1_df009b_1140106094390_990369_608' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1140106094390_329364_607' name='base$Class' visibility='private'
      UMLClass='_10_5_1_df009b_1140106056406_725870_600'
      association='_10_5_1_df009b_1140106094390_87790_606'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885343_144138_7929'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
        11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
          Metamodel::Classes::Kernel::Class' referentType='Class'/>
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140106107015_780843_611' name='Xprocess' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140106260359_319374_617' visibility='public'
    UMLClass='_10_5_1_df009b_1140106107015_780843_611'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140106260359_379916_618' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140106260359_760325_619' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_10_5_1_df009b_1140106260359_760325_619' name='extension$Xprocess'
      visibility='private' owningAssociation='_10_5_1_df009b_1140106260359_319374_617'
      association='_10_5_1_df009b_1140106260359_319374_617'
      type='_10_5_1_df009b_1140106107015_780843_611'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106260359_862246_621' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1140106260359_760325_619' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106260359_805577_620' visibility='public'
        owningLower='_10_5_1_df009b_1140106260359_760325_619' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1140106260359_379916_618' name='base$Class' visibility='private'
      UMLClass='_10_5_1_df009b_1140106107015_780843_611'
      association='_10_5_1_df009b_1140106260359_319374_617'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885343_144138_7929'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
        11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
          Metamodel::Classes::Kernel::Class' referentType='Class'/>
        </xmi:Extension>
      </type>
    </ownedAttribute>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1140176461171_331047_41' isReadOnly='true' name='self'
      visibility='private' UMLClass='_10_5_1_df009b_1140106107015_780843_611'
      type='_11_0_1_df009b_1147169951783_357389_163' />
  </ownedStereotype>

```

```

    <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147175350080_640007_213' isReadOnly='true' name='offspring'
visibility='private' UMLClass='_10_5_1_df009b_1140106107015_780843_611'
type='_11_0_1_df009b_1147169951783_357389_163'>/>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147175387798_580173_214' isReadOnly='true' name='parent'
visibility='private' UMLClass='_10_5_1_df009b_1140106107015_780843_611'
type='_11_0_1_df009b_1147169951783_357389_163'>/>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147175417564_124727_215' isReadOnly='true' name='sender'
visibility='private' UMLClass='_10_5_1_df009b_1140106107015_780843_611'
type='_11_0_1_df009b_1147169951783_357389_163'>/>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140106285359_369569_622' name='interface' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140106322906_317079_628' visibility='public'
UMLClass='_10_5_1_df009b_1140106285359_369569_622'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140106322906_83412_629'>/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140106322906_202439_630'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140106322906_202439_630' name='extension$interface'
visibility='private' owningAssociation='_10_5_1_df009b_1140106322906_317079_628'
association='_10_5_1_df009b_1140106322906_317079_628'
type='_10_5_1_df009b_1140106285359_369569_622'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106322906_128306_632' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140106322906_202439_630'>/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106322906_587101_631' visibility='public'
owningLower='_10_5_1_df009b_1140106322906_202439_630'>/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140106322906_83412_629' name='base$Interface'
visibility='private' UMLClass='_10_5_1_df009b_1140106285359_369569_622'
association='_10_5_1_df009b_1140106322906_317079_628'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884649_358727_7668'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Interfaces::Interface' referentType='Class'>/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146148946140_543463_259'
name='ownedSignals' visibility='public'
UMLClass='_10_5_1_df009b_1140106285359_369569_622'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'>/>
            </xmi:Extension>
        </type>
        <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146148972984_81430_261' value='*' visibility='public'
owningUpper='_11_0_df009b_1146148946140_543463_259'>/>
        <lowerValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146148972984_305080_260' value='*' visibility='public'
owningLower='_11_0_df009b_1146148946140_543463_259'>/>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140106336937_40295_633' name='primitive' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140106374890_915632_639' visibility='public'
UMLClass='_10_5_1_df009b_1140106336937_40295_633'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140106374890_697959_640'>/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140106374890_168365_641'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140106374890_168365_641' name='extension$primitiveType'
visibility='private' owningAssociation='_10_5_1_df009b_1140106374890_915632_639'

```

```

association='_10_5_1_df009b_1140106374890_915632_639'
type='_10_5_1_df009b_1140106336937_40295_633'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106374890_945293_643' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140106374890_168365_641'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106374890_525413_642' visibility='public'
owningLower='_10_5_1_df009b_1140106374890_168365_641'/>
</ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140106374890_697959_640' name='base$PrimitiveType'
visibility='private' UMLClass='_10_5_1_df009b_1140106336937_40295_633'
association='_10_5_1_df009b_1140106374890_915632_639'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885449_652048_7963'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
  <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::PrimitiveType' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140106405062_537810_644' name='dataType' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140106435921_671821_650' visibility='public'
UMLClass='_10_5_1_df009b_1140106405062_537810_644'>
  <memberEnd xmi:idref='_10_5_1_df009b_1140106435921_96527_651'/>
  <memberEnd xmi:idref='_10_5_1_df009b_1140106435921_993294_652'/>
  <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140106435921_993294_652' name='extension$dataType'
visibility='private' owningAssociation='_10_5_1_df009b_1140106435921_671821_650'
association='_10_5_1_df009b_1140106435921_671821_650'
type='_10_5_1_df009b_1140106405062_537810_644'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106435921_456327_654' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140106435921_993294_652'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106435921_657624_653' visibility='public'
owningLower='_10_5_1_df009b_1140106435921_993294_652'/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140106435921_96527_651' name='base$DataType'
visibility='private' UMLClass='_10_5_1_df009b_1140106405062_537810_644'
association='_10_5_1_df009b_1140106435921_671821_650'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
  <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140106523093_219683_655' name='operation' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140106553078_272117_661' visibility='public'
UMLClass='_10_5_1_df009b_1140106523093_219683_655'>
  <memberEnd xmi:idref='_10_5_1_df009b_1140106553078_340599_662'/>
  <memberEnd xmi:idref='_10_5_1_df009b_1140106553078_62531_663'/>
  <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140106553078_62531_663' name='extension$operation'
visibility='private' owningAssociation='_10_5_1_df009b_1140106553078_272117_661'
association='_10_5_1_df009b_1140106553078_272117_661'
type='_10_5_1_df009b_1140106523093_219683_655'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106553078_742573_665' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140106553078_62531_663'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106553078_358102_664' visibility='public'
owningLower='_10_5_1_df009b_1140106553078_62531_663'/>
  </ownedEnd>

```

```

</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140106553078_340599_662' name='base$Operation'
visibility='private' UMLClass='_10_5_1_df009b_1140106523093_219683_655'
association='_10_5_1_df009b_1140106553078_272117_661'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884698_645168_7692'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Operation' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140106621984_350093_666' name='port' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140106646781_215807_672' visibility='public'
UMLClass='_10_5_1_df009b_1140106621984_350093_666'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140106646781_631783_673'/>
    <memberEnd xmi:idref='_10_5_1_df009b_1140106646781_883640_674'/>
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140106646781_883640_674' name='extension$port'
visibility='private' owningAssociation='_10_5_1_df009b_1140106646781_215807_672'
association='_10_5_1_df009b_1140106646781_215807_672'
type='_10_5_1_df009b_1140106621984_350093_666'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106646781_430313_676' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140106646781_883640_674'/>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140106646781_194101_675' visibility='public'
owningLower='_10_5_1_df009b_1140106646781_883640_674'/>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140106646781_631783_673' name='base$Port' visibility='private'
UMLClass='_10_5_1_df009b_1140106621984_350093_666'
association='_10_5_1_df009b_1140106646781_215807_672'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704911199_900094_9269'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CompositeStructures::Ports::Port' referentType='Class'/>
    </xmi:Extension>
  </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146145746437_351201_174'
name='queueDiscipline' visibility='public'
UMLClass='_10_5_1_df009b_1140106621984_350093_666'
type='_11_0_df009b_1146146279515_868848_186'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146306390_276455_188' value='1' visibility='public'
owningUpper='_11_0_df009b_1146145746437_351201_174'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146306390_821673_187' visibility='public'
owningLower='_11_0_df009b_1146145746437_351201_174'/>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146314187_389828_189'
name='isDynamic' visibility='public'
UMLClass='_10_5_1_df009b_1140106621984_350093_666'>
  <type xmi:type='uml:DataType'
href='UML_Standard_Profile.xml|eee_1045467100323_191782_59'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML Standard
Profile::datatypes::boolean' referentType='DataType'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146607140_441523_201'
isReadOnly='true' isStatic='true' name='instances' visibility='public'
UMLClass='_10_5_1_df009b_1140106621984_350093_666'
type='_10_5_1_df009b_1141226913868_167015_565'/>
</ownedStereotype>

```

```

<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140106698671_827652_677' name='inherits' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140106737593_669480_683' visibility='public'
    UMLClass='_10_5_1_df009b_1140106698671_827652_677'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140106737593_769714_684' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140106737593_204977_685' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_10_5_1_df009b_1140106737593_204977_685' name='extension$inherits'
      visibility='private' owningAssociation='_10_5_1_df009b_1140106737593_669480_683'
      association='_10_5_1_df009b_1140106737593_669480_683'
      type='_10_5_1_df009b_1140106698671_827652_677'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106737593_460436_687' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1140106737593_204977_685' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106737593_961224_686' visibility='public'
        owningLower='_10_5_1_df009b_1140106737593_204977_685' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1140106737593_769714_684' name='base$Generalization'
      visibility='private' UMLClass='_10_5_1_df009b_1140106698671_827652_677'
      association='_10_5_1_df009b_1140106737593_669480_683'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885195_432731_7879'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Classes::Kernel::Generalization' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140106755296_318312_688' name='generalization'
  visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140106779765_482395_694' visibility='public'
    UMLClass='_10_5_1_df009b_1140106755296_318312_688'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140106779781_349202_695' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140106779781_632592_696' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_10_5_1_df009b_1140106779781_632592_696' name='extension$generalization'
      visibility='private' owningAssociation='_10_5_1_df009b_1140106779765_482395_694'
      association='_10_5_1_df009b_1140106779765_482395_694'
      type='_10_5_1_df009b_1140106755296_318312_688'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106779781_771026_698' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1140106779781_632592_696' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140106779781_755906_697' visibility='public'
        owningLower='_10_5_1_df009b_1140106779781_632592_696' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1140106779781_349202_695' name='base$Generalization'
      visibility='private' UMLClass='_10_5_1_df009b_1140106755296_318312_688'
      association='_10_5_1_df009b_1140106779765_482395_694'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885195_432731_7879'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Classes::Kernel::Generalization' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140106941796_32988_699' name='channel' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140107040140_545686_705' visibility='public'
    UMLClass='_10_5_1_df009b_1140106941796_32988_699'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140107040140_900467_706' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140107040140_894255_707' />

```

```

    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140107040140_894255_707' name='extension$channel'
visibility='private' owningAssociation='_10_5_1_df009b_1140107040140_545686_705'
association='_10_5_1_df009b_1140107040140_545686_705'
type='_10_5_1_df009b_1140106941796_32988_699'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107040140_88284_709' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140107040140_894255_707'>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107040140_63104_708' visibility='public'
owningLower='_10_5_1_df009b_1140107040140_894255_707'>
    </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140107040140_900467_706' name='base$Connector'
visibility='private' UMLClass='_10_5_1_df009b_1140106941796_32988_699'
association='_10_5_1_df009b_1140107040140_545686_705'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884559_152470_7636'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CompositeStructures::InternalStructures::Connector' referentType='Class'>
        </xmi:Extension>
    </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146151564093_965644_276'
name='signalList0' visibility='public'
UMLClass='_10_5_1_df009b_1140106941796_32988_699'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'>
        </xmi:Extension>
    </type>
    <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146151590531_357655_280' value='*' visibility='public'
owningUpper='_11_0_df009b_1146151564093_965644_276'>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146151590531_508370_279' visibility='public'
owningLower='_11_0_df009b_1146151564093_965644_276'>
    </ownedAttribute>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146151596406_808212_281'
name='signalList1' visibility='public'
UMLClass='_10_5_1_df009b_1140106941796_32988_699'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'>
        </xmi:Extension>
    </type>
    <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146151612875_498587_283' value='*' visibility='public'
owningUpper='_11_0_df009b_1146151596406_808212_281'>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146151612875_985152_282' visibility='public'
owningLower='_11_0_df009b_1146151596406_808212_281'>
    </ownedAttribute>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146151696437_226669_284'
name='delay' visibility='public' UMLClass='_10_5_1_df009b_1140106941796_32988_699'>
    <type xmi:type='uml:DataType'
href='UML_Standard_Profile.xml|eee_1045467100323_191782_59'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML Standard
Profile::datatypes::boolean' referentType='DataType'>
        </xmi:Extension>
    </type>
    <defaultValue xmi:type='uml:LiteralBoolean'
xmi:id='_11_0_df009b_1146151707484_685166_285' visibility='public'
owningProperty='_11_0_df009b_1146151696437_226669_284'>
    </ownedAttribute>

```



```

    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146151718312_534960_286'
name='distinctSignals' visibility='public'
UMLClass='_10_5_1_df009b_1140106941796_32988_699'>
    <type xmi:type='uml:DataType'
href='UML_Standard_Profile.xml|eee_1045467100323_191782_59'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML Standard
Profile::datatypes::boolean' referentType='DataType'/>
    </xmi:Extension>
    </type>
    <defaultValue xmi:type='uml:LiteralBoolean'
xmi:id='_11_0_df009b_1146151733359_153603_287' visibility='public'
owningProperty='_11_0_df009b_1146151718312_534960_286'/>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140107089234_793685_710' name='enumeration'
visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140107116515_294276_716' visibility='public'
UMLClass='_10_5_1_df009b_1140107089234_793685_710'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140107116515_500720_717'/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140107116515_173442_718'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140107116515_173442_718' name='extension$enumeration'
visibility='private' owningAssociation='_10_5_1_df009b_1140107116515_294276_716'
association='_10_5_1_df009b_1140107116515_294276_716'
type='_10_5_1_df009b_1140107089234_793685_710'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107116515_903317_720' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140107116515_173442_718'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107116515_524764_719' visibility='public'
owningLower='_10_5_1_df009b_1140107116515_173442_718'/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140107116515_500720_717' name='base$Enumeration'
visibility='private' UMLClass='_10_5_1_df009b_1140107089234_793685_710'
association='_10_5_1_df009b_1140107116515_294276_716'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885400_895774_7947'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Enumeration' referentType='Class'/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140107137250_118848_721' name='package' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140107158421_614770_727' visibility='public'
UMLClass='_10_5_1_df009b_1140107137250_118848_721'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140107158421_98340_728'/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140107158421_769704_729'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140107158421_769704_729' name='extension$package'
visibility='private' owningAssociation='_10_5_1_df009b_1140107158421_614770_727'
association='_10_5_1_df009b_1140107158421_614770_727'
type='_10_5_1_df009b_1140107137250_118848_721'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107158421_407170_731' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140107158421_769704_729'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107158421_180458_730' visibility='public'
owningLower='_10_5_1_df009b_1140107158421_769704_729'/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140107158421_98340_728' name='base$Package'
visibility='private' UMLClass='_10_5_1_df009b_1140107137250_118848_721'
association='_10_5_1_df009b_1140107158421_614770_727'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885298_713292_7913'>

```

```

    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Package' referentType='Class'/>
    </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140107200765_961902_732' name='timer' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140107227703_236617_738' visibility='public'
UMLClass='_10_5_1_df009b_1140107200765_961902_732'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140107227703_464991_739' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140107227718_45354_740' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140107227718_45354_740' name='extension$timer'
visibility='private' owningAssociation='_10_5_1_df009b_1140107227703_236617_738'
association='_10_5_1_df009b_1140107227703_236617_738'
type='_10_5_1_df009b_1140107200765_961902_732'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107227718_945877_742' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140107227718_45354_740' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107227718_678216_741' visibility='public'
owningLower='_10_5_1_df009b_1140107227718_45354_740' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140107227703_464991_739' name='base$Signal'
visibility='private' UMLClass='_10_5_1_df009b_1140107200765_961902_732'
association='_10_5_1_df009b_1140107227703_236617_738'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140107751718_840616_743' name='class' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140107787296_215895_749' visibility='public'
UMLClass='_10_5_1_df009b_1140107751718_840616_743'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140107787296_258841_750' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140107787296_29545_751' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140107787296_29545_751' name='extension$class'
visibility='private' owningAssociation='_10_5_1_df009b_1140107787296_215895_749'
association='_10_5_1_df009b_1140107787296_215895_749'
type='_10_5_1_df009b_1140107751718_840616_743'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107787296_502384_753' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140107787296_29545_751' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140107787296_734129_752' visibility='public'
owningLower='_10_5_1_df009b_1140107787296_29545_751' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140107787296_258841_750' name='base$Class' visibility='private'
UMLClass='_10_5_1_df009b_1140107751718_840616_743'
association='_10_5_1_df009b_1140107787296_215895_749'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885343_144138_7929'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Class' referentType='Class'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>

```

```

<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140176192578_318697_1' name='activity' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140176216296_33976_7' visibility='public'
    UMLClass='_10_5_1_df009b_1140176192578_318697_1'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140176216296_515424_8' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140176216296_14030_9' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_10_5_1_df009b_1140176216296_14030_9'
      name='extension$activity' visibility='private'
      owningAssociation='_10_5_1_df009b_1140176216296_33976_7'
      association='_10_5_1_df009b_1140176216296_33976_7'
      type='_10_5_1_df009b_1140176192578_318697_1'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140176216296_462657_11' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1140176216296_14030_9' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140176216296_934413_10' visibility='public'
        owningLower='_10_5_1_df009b_1140176216296_14030_9' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_10_5_1_df009b_1140176216296_515424_8'
      name='base$Activity' visibility='private'
      UMLClass='_10_5_1_df009b_1140176192578_318697_1'
      association='_10_5_1_df009b_1140176216296_33976_7'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704892254_121736_8466'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Activities::FundamentalActivities::Activity' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140176226250_292958_12' name='sequenceNode'
  visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140176253687_61759_18' visibility='public'
    UMLClass='_10_5_1_df009b_1140176226250_292958_12'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140176253687_290890_19' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140176253687_53445_20' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_10_5_1_df009b_1140176253687_53445_20'
      name='extension$sequenceNode' visibility='private'
      owningAssociation='_10_5_1_df009b_1140176253687_61759_18'
      association='_10_5_1_df009b_1140176253687_61759_18'
      type='_10_5_1_df009b_1140176226250_292958_12'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140176253687_33916_22' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1140176253687_53445_20' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1140176253687_834068_21' visibility='public'
        owningLower='_10_5_1_df009b_1140176253687_53445_20' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1140176253687_290890_19' name='base$SequenceNode'
      visibility='private' UMLClass='_10_5_1_df009b_1140176226250_292958_12'
      association='_10_5_1_df009b_1140176253687_61759_18'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704892401_738929_8490'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Activities::StructuredActivities::SequenceNode' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1140176264593_624011_23' name='output' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1140176283531_248975_29' visibility='public'
    UMLClass='_10_5_1_df009b_1140176264593_624011_23'>
    <memberEnd xmi:idref='_10_5_1_df009b_1140176283531_949714_30' />
    <memberEnd xmi:idref='_10_5_1_df009b_1140176283531_135123_31' />

```

```

    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140176283531_135123_31' name='extension$output'
visibility='private' owningAssociation='_10_5_1_df009b_1140176283531_248975_29'
association='_10_5_1_df009b_1140176283531_248975_29'
type='_10_5_1_df009b_1140176264593_624011_23'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140176283531_13680_33' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140176283531_135123_31'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140176283531_930633_32' visibility='public'
owningLower='_10_5_1_df009b_1140176283531_135123_31'/>
    </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140176283531_949714_30' name='base$SendSignalAction'
visibility='private' UMLClass='_10_5_1_df009b_1140176264593_624011_23'
association='_10_5_1_df009b_1140176283531_248975_29'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704887121_121073_8157'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::BasicActions::SendSignalAction' referentType='Class'/>
    </xmi:Extension>
    </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140176318406_320069_34' name='via' visibility='private'
UMLClass='_10_5_1_df009b_1140176264593_624011_23'
type='_10_5_1_df009b_1140106621984_350093_666'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140176665390_585437_160' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140176318406_320069_34'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140176665390_305394_159' visibility='public'
owningLower='_10_5_1_df009b_1140176318406_320069_34'/>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147174209892_705097_210' name='attributes' visibility='public'
UMLClass='_10_5_1_df009b_1140176264593_624011_23'>
    <type xmi:type='uml:PrimitiveType'
href='UML_Standard_Profile.xml|_9_0_2_91a0295_1110274713995_297054_0'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML Standard
Profile::datatypes::String' referentType='PrimitiveType'/>
    </xmi:Extension>
    </type>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147175647673_695989_217' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147174209892_705097_210'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147175647673_826562_216' visibility='public'
owningLower='_11_0_1_df009b_1147174209892_705097_210'/>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140176423046_579854_35' name='pid' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147102960796_324296_6' visibility='public'
UMLClass='_10_5_1_df009b_1140176423046_579854_35'>
        <memberEnd xmi:idref='_11_0_1_df009b_1147102960796_952114_7'/>
        <memberEnd xmi:idref='_11_0_1_df009b_1147102960796_315690_8'/>
        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_1_df009b_1147102960796_315690_8'
name='extension$PId' visibility='private'
owningAssociation='_11_0_1_df009b_1147102960796_324296_6'
association='_11_0_1_df009b_1147102960796_324296_6'
type='_10_5_1_df009b_1140176423046_579854_35'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147102960812_570189_10' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147102960796_315690_8'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147102960796_787571_9' visibility='public'
owningLower='_11_0_1_df009b_1147102960796_315690_8'/>
        </ownedEnd>
    </nestedClassifier>

```

```

    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_1_df009b_1147102960796_952114_7'
name='base$DataType' visibility='private'
UMLClass='_10_5_1_df009b_1140176423046_579854_35'
association='_11_0_1_df009b_1147102960796_324296_6'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'/>
        </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140177817453_830794_172' name='begin' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140177840625_823948_178' visibility='public'
UMLClass='_10_5_1_df009b_1140177817453_830794_172'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140177840625_44417_179'/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140177840625_457194_180'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140177840625_457194_180' name='extension$begin'
visibility='private' owningAssociation='_10_5_1_df009b_1140177840625_823948_178'
association='_10_5_1_df009b_1140177840625_823948_178'
type='_10_5_1_df009b_1140177817453_830794_172'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140177840625_391221_182' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140177840625_457194_180'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140177840625_577146_181' visibility='public'
owningLower='_10_5_1_df009b_1140177840625_457194_180'/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140177840625_44417_179' name='base$InitialNode'
visibility='private' UMLClass='_10_5_1_df009b_1140177817453_830794_172'
association='_10_5_1_df009b_1140177840625_823948_178'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704893447_508814_8555'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::BasicActivities::InitialNode' referentType='Class'/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1140177848968_54313_183' name='return' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1140177880281_76051_189' visibility='public'
UMLClass='_10_5_1_df009b_1140177848968_54313_183'>
        <memberEnd xmi:idref='_10_5_1_df009b_1140177880281_921197_190'/>
        <memberEnd xmi:idref='_10_5_1_df009b_1140177880281_994569_191'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1140177880281_994569_191' name='extension$return'
visibility='private' owningAssociation='_10_5_1_df009b_1140177880281_76051_189'
association='_10_5_1_df009b_1140177880281_76051_189'
type='_10_5_1_df009b_1140177848968_54313_183'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140177880281_74745_193' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1140177880281_994569_191'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1140177880281_567560_192' visibility='public'
owningLower='_10_5_1_df009b_1140177880281_994569_191'/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1140177880281_921197_190' name='base$ActivityFinalNode'
visibility='private' UMLClass='_10_5_1_df009b_1140177848968_54313_183'
association='_10_5_1_df009b_1140177880281_76051_189'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704893490_637380_8563'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>

```

```

        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::BasicActivities::ActivityFinalNode' referentType='Class'/>
    </xmi:Extension>
    </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1141153940034_787730_34' name='task' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1141155037987_41942_236' visibility='public'
UMLClass='_10_5_1_df009b_1141153940034_787730_34'>
        <memberEnd xmi:idref='_10_5_1_df009b_1141155037987_597336_237'/>
        <memberEnd xmi:idref='_10_5_1_df009b_1141155037987_378790_238'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1141155037987_378790_238' name='extension$task'
visibility='private' owningAssociation='_10_5_1_df009b_1141155037987_41942_236'
association='_10_5_1_df009b_1141155037987_41942_236'
type='_10_5_1_df009b_1141153940034_787730_34'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1141155038003_460916_240' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1141155037987_378790_238'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1141155037987_424075_239' visibility='public'
owningLower='_10_5_1_df009b_1141155037987_378790_238'/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1141155037987_597336_237' name='base$OpaqueAction'
visibility='private' UMLClass='_10_5_1_df009b_1141153940034_787730_34'
association='_10_5_1_df009b_1141155037987_41942_236'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704886256_729983_8101'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::BasicActions::OpaqueAction' referentType='Class'/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1141154781956_559817_225' name='decisionNode'
visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_10_5_1_df009b_1141216620978_831938_496' visibility='public'
UMLClass='_10_5_1_df009b_1141154781956_559817_225'>
        <memberEnd xmi:idref='_10_5_1_df009b_1141216620978_370178_497'/>
        <memberEnd xmi:idref='_10_5_1_df009b_1141216620978_489859_498'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_10_5_1_df009b_1141216620978_489859_498' name='extension$decisionNode'
visibility='private' owningAssociation='_10_5_1_df009b_1141216620978_831938_496'
association='_10_5_1_df009b_1141216620978_831938_496'
type='_10_5_1_df009b_1141154781956_559817_225'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1141216620978_240888_500' value='1' visibility='public'
owningUpper='_10_5_1_df009b_1141216620978_489859_498'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_10_5_1_df009b_1141216620978_327511_499' visibility='public'
owningLower='_10_5_1_df009b_1141216620978_489859_498'/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
xmi:id='_10_5_1_df009b_1141216620978_370178_497' name='base$DecisionNode'
visibility='private' UMLClass='_10_5_1_df009b_1141154781956_559817_225'
association='_10_5_1_df009b_1141216620978_831938_496'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704921751_161098_9437'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::IntermediateActivities::DecisionNode' referentType='Class'/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>

```

```

<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1141216459618_489462_483' name='mergeNode' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1141216477118_207176_489' visibility='public'
    UMLClass='_10_5_1_df009b_1141216459618_489462_483'>
    <memberEnd xmi:idref='_10_5_1_df009b_1141216477118_522381_490' />
    <memberEnd xmi:idref='_10_5_1_df009b_1141216477118_251882_491' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_10_5_1_df009b_1141216477118_251882_491' name='extension$mergeNode'
      visibility='private' owningAssociation='_10_5_1_df009b_1141216477118_207176_489'
      association='_10_5_1_df009b_1141216477118_207176_489'
      type='_10_5_1_df009b_1141216459618_489462_483'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1141216477118_1554_493' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1141216477118_251882_491' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1141216477118_771634_492' visibility='public'
        owningLower='_10_5_1_df009b_1141216477118_251882_491' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1141216477118_522381_490' name='base$MergeNode'
      visibility='private' UMLClass='_10_5_1_df009b_1141216459618_489462_483'
      association='_10_5_1_df009b_1141216477118_207176_489'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704921684_282475_9429'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Activities::IntermediateActivities::MergeNode' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1141221936915_842382_508' name='controlFlow'
  visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_10_5_1_df009b_1141221953774_374565_514' visibility='public'
    UMLClass='_10_5_1_df009b_1141221936915_842382_508'>
    <memberEnd xmi:idref='_10_5_1_df009b_1141221953774_393372_515' />
    <memberEnd xmi:idref='_10_5_1_df009b_1141221953774_254229_516' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_10_5_1_df009b_1141221953774_254229_516' name='extension$controlFlow'
      visibility='private' owningAssociation='_10_5_1_df009b_1141221953774_374565_514'
      association='_10_5_1_df009b_1141221953774_374565_514'
      type='_10_5_1_df009b_1141221936915_842382_508'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1141221953774_774569_518' value='1' visibility='public'
        owningUpper='_10_5_1_df009b_1141221953774_254229_516' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_10_5_1_df009b_1141221953774_682445_517' visibility='public'
        owningLower='_10_5_1_df009b_1141221953774_254229_516' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_10_5_1_df009b_1141221953774_393372_515' name='base$ControlFlow'
      visibility='private' UMLClass='_10_5_1_df009b_1141221936915_842382_508'
      association='_10_5_1_df009b_1141221953774_374565_514'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704893364_624946_8539'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Activities::BasicActivities::ControlFlow' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_10_5_1_df009b_1141226913868_167015_565' name='natural' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_11_0_1_df009b_1147103016187_492389_21' visibility='public'
    UMLClass='_10_5_1_df009b_1141226913868_167015_565'>
    <memberEnd xmi:idref='_11_0_1_df009b_1147103016187_399909_22' />
    <memberEnd xmi:idref='_11_0_1_df009b_1147103016187_214604_23' />

```

```

    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147103016187_214604_23' name='extension$natural'
visibility='private' owningAssociation='_11_0_1_df009b_1147103016187_492389_21'
association='_11_0_1_df009b_1147103016187_492389_21'
type='_10_5_1_df009b_1141226913868_167015_565'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147103016187_103603_25' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147103016187_214604_23'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147103016187_575627_24' visibility='public'
owningLower='_11_0_1_df009b_1147103016187_214604_23'/>
    </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147103016187_399909_22' name='base$DataType'
visibility='private' UMLClass='_10_5_1_df009b_1141226913868_167015_565'
association='_11_0_1_df009b_1147103016187_492389_21'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'/>
    </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_10_5_1_df009b_1141226929978_616494_571' name='integer' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147103000734_375860_16' visibility='public'
UMLClass='_10_5_1_df009b_1141226929978_616494_571'>
        <memberEnd xmi:idref='_11_0_1_df009b_1147103000734_297592_17'/>
        <memberEnd xmi:idref='_11_0_1_df009b_1147103000734_546207_18'/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147103000734_546207_18' name='extension$integer'
visibility='private' owningAssociation='_11_0_1_df009b_1147103000734_375860_16'
association='_11_0_1_df009b_1147103000734_375860_16'
type='_10_5_1_df009b_1141226929978_616494_571'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147103000734_871518_20' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147103000734_546207_18'/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147103000734_254083_19' visibility='public'
owningLower='_11_0_1_df009b_1147103000734_546207_18'/>
            </ownedEnd>
        </nestedClassifier>
        <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147103000734_297592_17' name='base$DataType'
visibility='private' UMLClass='_10_5_1_df009b_1141226929978_616494_571'
association='_11_0_1_df009b_1147103000734_375860_16'>
            <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
                <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'/>
                </xmi:Extension>
            </type>
        </ownedAttribute>
    </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146059277799_859679_126' name='operationCall'
visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146059304674_107388_132' visibility='public'
UMLClass='_11_0_df009b_1146059277799_859679_126'>
        <memberEnd xmi:idref='_11_0_df009b_1146059304674_115396_133'/>
        <memberEnd xmi:idref='_11_0_df009b_1146059304674_878665_134'/>
        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146059304674_878665_134'
name='extension$methodCall' visibility='private'
owningAssociation='_11_0_df009b_1146059304674_107388_132'
association='_11_0_df009b_1146059304674_107388_132'
type='_11_0_df009b_1146059277799_859679_126'>
    </ownedEnd>
    </nestedClassifier>
</ownedStereotype>

```



```

    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059304674_992564_136' value='1' visibility='public'
owningUpper='_11_0_df009b_1146059304674_878665_134'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059304674_900569_135' visibility='public'
owningLower='_11_0_df009b_1146059304674_878665_134'/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146059304674_115396_133'
name='base$CallOperationAction' visibility='private'
UMLClass='_11_0_df009b_1146059277799_859679_126'
association='_11_0_df009b_1146059304674_107388_132'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704887160_569760_8165'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::BasicActions::CallOperationAction' referentType='Class'/>
    </xmi:Extension>
  </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1148906066828_11753_118' name='via' visibility='public'
UMLClass='_11_0_df009b_1146059277799_859679_126'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704911199_900094_9269'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CompositeStructures::Ports::Port' referentType='Class'/>
    </xmi:Extension>
  </type>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1148906085671_483009_120' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1148906066828_11753_118'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1148906085671_289606_119' visibility='public'
owningLower='_11_0_1_df009b_1148906066828_11753_118'/>
  </ownedAttribute>
  <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1148906092531_49812_121' isOrdered='true' name='attributes'
visibility='public' UMLClass='_11_0_df009b_1146059277799_859679_126'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884574_96724_7644'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Property' referentType='Class'/>
    </xmi:Extension>
  </type>
  <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_1_df009b_1148906131531_211057_123' value='*' visibility='public'
owningUpper='_11_0_1_df009b_1148906092531_49812_121'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1148906131531_98929_122' visibility='public'
owningLower='_11_0_1_df009b_1148906092531_49812_121'/>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146059848862_29125_149'
name='for' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146059863658_54804_155' visibility='public'
UMLClass='_11_0_df009b_1146059848862_29125_149'>
    <memberEnd xmi:idref='_11_0_df009b_1146059863658_633875_156'/>
    <memberEnd xmi:idref='_11_0_df009b_1146059863658_601682_157'/>
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146059863658_601682_157'
name='extension$for' visibility='private'
owningAssociation='_11_0_df009b_1146059863658_54804_155'
association='_11_0_df009b_1146059863658_54804_155'
type='_11_0_df009b_1146059848862_29125_149'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059863658_928871_159' value='1' visibility='public'
owningUpper='_11_0_df009b_1146059863658_601682_157'/>
        <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059863658_345327_158' visibility='public'
owningLower='_11_0_df009b_1146059863658_601682_157'/>
      </ownedEnd>

```

```

</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146059863658_633875_156'
name='base$LoopNode' visibility='private'
UMLClass='_11_0_df009b_1146059848862_29125_149'
association='_11_0_df009b_1146059863658_54804_155'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704892169_230137_8450'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::StructuredActivities::LoopNode' referentType='Class' />
    </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146059876190_408081_160' name='while' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146059892971_291174_166' visibility='public'
UMLClass='_11_0_df009b_1146059876190_408081_160'>
    <memberEnd xmi:idref='_11_0_df009b_1146059892971_73713_167' />
    <memberEnd xmi:idref='_11_0_df009b_1146059892971_815899_168' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146059892971_815899_168'
name='extension$while' visibility='private'
owningAssociation='_11_0_df009b_1146059892971_291174_166'
association='_11_0_df009b_1146059892971_291174_166'
type='_11_0_df009b_1146059876190_408081_160'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059892971_323458_170' value='1' visibility='public'
owningUpper='_11_0_df009b_1146059892971_815899_168' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059892971_394489_169' visibility='public'
owningLower='_11_0_df009b_1146059892971_815899_168' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146059892971_73713_167'
name='base$LoopNode' visibility='private'
UMLClass='_11_0_df009b_1146059876190_408081_160'
association='_11_0_df009b_1146059892971_291174_166'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704892169_230137_8450'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::StructuredActivities::LoopNode' referentType='Class' />
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146059939205_21895_171'
name='repeat' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146059965002_115553_177' visibility='public'
UMLClass='_11_0_df009b_1146059939205_21895_171'>
    <memberEnd xmi:idref='_11_0_df009b_1146059965002_819499_178' />
    <memberEnd xmi:idref='_11_0_df009b_1146059965002_251118_179' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146059965002_251118_179'
name='extension$repeat' visibility='private'
owningAssociation='_11_0_df009b_1146059965002_115553_177'
association='_11_0_df009b_1146059965002_115553_177'
type='_11_0_df009b_1146059939205_21895_171'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059965002_682582_181' value='1' visibility='public'
owningUpper='_11_0_df009b_1146059965002_251118_179' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059965002_332873_180' visibility='public'
owningLower='_11_0_df009b_1146059965002_251118_179' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146059965002_819499_178'
name='base$LoopNode' visibility='private'
UMLClass='_11_0_df009b_1146059939205_21895_171'
association='_11_0_df009b_1146059965002_115553_177'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704892169_230137_8450'>

```

```

    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::StructuredActivities::LoopNode' referentType='Class' />
    </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146059982502_75624_182'
name='noOperation' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146059999924_463235_188' visibility='public'
UMLClass='_11_0_df009b_1146059982502_75624_182'>
    <memberEnd xmi:idref='_11_0_df009b_1146059999924_910075_189' />
    <memberEnd xmi:idref='_11_0_df009b_1146059999924_669167_190' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146059999924_669167_190'
name='extension$noOperation' visibility='private'
owningAssociation='_11_0_df009b_1146059999924_463235_188'
association='_11_0_df009b_1146059999924_463235_188'
type='_11_0_df009b_1146059982502_75624_182'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059999924_422104_192' value='1' visibility='public'
owningUpper='_11_0_df009b_1146059999924_669167_190' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146059999924_451399_191' visibility='public'
owningLower='_11_0_df009b_1146059999924_669167_190' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146059999924_910075_189'
name='base$OpaqueAction' visibility='private'
UMLClass='_11_0_df009b_1146059982502_75624_182'
association='_11_0_df009b_1146059999924_463235_188'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704886256_729983_8101'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::BasicActions::OpaqueAction' referentType='Class' />
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146060059971_887823_193' name='continue' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146060082862_157436_199' visibility='public'
UMLClass='_11_0_df009b_1146060059971_887823_193'>
    <memberEnd xmi:idref='_11_0_df009b_1146060082862_862336_200' />
    <memberEnd xmi:idref='_11_0_df009b_1146060082862_445876_201' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146060082862_445876_201'
name='extension$continue' visibility='private'
owningAssociation='_11_0_df009b_1146060082862_157436_199'
association='_11_0_df009b_1146060082862_157436_199'
type='_11_0_df009b_1146060059971_887823_193'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146060082862_775717_203' value='1' visibility='public'
owningUpper='_11_0_df009b_1146060082862_445876_201' />
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146060082862_425046_202' visibility='public'
owningLower='_11_0_df009b_1146060082862_445876_201' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146060082862_862336_200'
name='base$OpaqueAction' visibility='private'
UMLClass='_11_0_df009b_1146060059971_887823_193'
association='_11_0_df009b_1146060082862_157436_199'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704886256_729983_8101'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::BasicActions::OpaqueAction' referentType='Class' />
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>

```

```

<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_11_0_df009b_1146060145158_949303_204' name='break' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_11_0_df009b_1146060161877_436215_210' visibility='public'
    UMLClass='_11_0_df009b_1146060145158_949303_204'>
    <memberEnd xmi:idref='_11_0_df009b_1146060161877_797755_211' />
    <memberEnd xmi:idref='_11_0_df009b_1146060161877_456779_212' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146060161877_456779_212'
      name='extension$break' visibility='private'
      owningAssociation='_11_0_df009b_1146060161877_436215_210'
      association='_11_0_df009b_1146060161877_436215_210'
      type='_11_0_df009b_1146060145158_949303_204'>
    <upperValue xmi:type='uml:LiteralInteger'
      xmi:id='_11_0_df009b_1146060161877_448717_214' value='1' visibility='public'
      owningUpper='_11_0_df009b_1146060161877_456779_212' />
    <lowerValue xmi:type='uml:LiteralInteger'
      xmi:id='_11_0_df009b_1146060161877_825309_213' visibility='public'
      owningLower='_11_0_df009b_1146060161877_456779_212' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146060161877_797755_211'
    name='base$OpaqueAction' visibility='private'
    UMLClass='_11_0_df009b_1146060145158_949303_204'
    association='_11_0_df009b_1146060161877_436215_210'>
    <type xmi:type='uml:Class'
      href='UML_Standard_Profile.xml|_9_0_62a020a_1105704886256_729983_8101'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
      11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
      Metamodel::Actions::BasicActions::OpaqueAction' referentType='Class' />
    </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146138182187_605610_12'
  name='union' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_11_0_df009b_1146138202000_673864_18' visibility='public'
    UMLClass='_11_0_df009b_1146138182187_605610_12'>
    <memberEnd xmi:idref='_11_0_df009b_1146138202000_3877_19' />
    <memberEnd xmi:idref='_11_0_df009b_1146138202000_199996_20' />
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146138202000_199996_20'
      name='extension$union' visibility='private'
      owningAssociation='_11_0_df009b_1146138202000_673864_18'
      association='_11_0_df009b_1146138202000_673864_18'
      type='_11_0_df009b_1146138182187_605610_12'>
    <upperValue xmi:type='uml:LiteralInteger'
      xmi:id='_11_0_df009b_1146138202000_131744_22' value='1' visibility='public'
      owningUpper='_11_0_df009b_1146138202000_199996_20' />
    <lowerValue xmi:type='uml:LiteralInteger'
      xmi:id='_11_0_df009b_1146138202000_540657_21' visibility='public'
      owningLower='_11_0_df009b_1146138202000_199996_20' />
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146138202000_3877_19'
    name='base$DataType' visibility='private'
    UMLClass='_11_0_df009b_1146138182187_605610_12'
    association='_11_0_df009b_1146138202000_673864_18'>
    <type xmi:type='uml:Class'
      href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
      11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
      Metamodel::Classes::Kernel::DataType' referentType='Class' />
    </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146138210828_909650_23'
  name='value' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_11_0_1_df009b_1147169791251_757155_158' visibility='public'
    UMLClass='_11_0_df009b_1146138210828_909650_23'>
    <memberEnd xmi:idref='_11_0_1_df009b_1147169791251_193377_159' />
    <memberEnd xmi:idref='_11_0_1_df009b_1147169791251_113788_160' />

```

```

    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147169791251_113788_160' name='extension$value'
visibility='private' owningAssociation='_11_0_1_df009b_1147169791251_757155_158'
association='_11_0_1_df009b_1147169791251_757155_158'
type='_11_0_df009b_1146138210828_909650_23'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169791251_359770_162' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147169791251_113788_160'>/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169791251_298610_161' visibility='public'
owningLower='_11_0_1_df009b_1147169791251_113788_160'>/>
    </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147169791251_193377_159' name='base$Property'
visibility='private' UMLClass='_11_0_df009b_1146138210828_909650_23'
association='_11_0_1_df009b_1147169791251_757155_158'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884574_96724_7644'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Property' referentType='Class'>/>
        </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146138350109_994569_155' name='object' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147169767142_31535_153' visibility='public'
UMLClass='_11_0_df009b_1146138350109_994569_155'>
        <memberEnd xmi:idref='_11_0_1_df009b_1147169767142_459003_154'>/>
        <memberEnd xmi:idref='_11_0_1_df009b_1147169767142_474780_155'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147169767142_474780_155' name='extension$object'
visibility='private' owningAssociation='_11_0_1_df009b_1147169767142_31535_153'
association='_11_0_1_df009b_1147169767142_31535_153'
type='_11_0_df009b_1146138350109_994569_155'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169767142_300735_157' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147169767142_474780_155'>/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169767142_843068_156' visibility='public'
owningLower='_11_0_1_df009b_1147169767142_474780_155'>/>
            </ownedEnd>
        </nestedClassifier>
        <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147169767142_459003_154' name='base$Property'
visibility='private' UMLClass='_11_0_df009b_1146138350109_994569_155'
association='_11_0_1_df009b_1147169767142_31535_153'>
            <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884574_96724_7644'>
                <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Property' referentType='Class'>/>
                </xmi:Extension>
            </type>
        </ownedAttribute>
    </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146146360328_775013_190' name='boolean' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147102984640_902088_11' visibility='public'
UMLClass='_11_0_df009b_1146146360328_775013_190'>
        <memberEnd xmi:idref='_11_0_1_df009b_1147102984640_572301_12'>/>
        <memberEnd xmi:idref='_11_0_1_df009b_1147102984640_463300_13'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147102984640_463300_13' name='extension$boolean'
visibility='private' owningAssociation='_11_0_1_df009b_1147102984640_902088_11'
association='_11_0_1_df009b_1147102984640_902088_11'
type='_11_0_df009b_1146146360328_775013_190'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147102984640_114355_15' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147102984640_463300_13'>/>

```

```

    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147102984640_726524_14' visibility='public'
owningLower='_11_0_1_df009b_1147102984640_463300_13'/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147102984640_572301_12' name='base$DataType'
visibility='private' UMLClass='_11_0_df009b_1146146360328_775013_190'
association='_11_0_1_df009b_1147102984640_902088_11'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'>
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146146676640_543843_202' name='setTimer' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146146698031_316325_208' visibility='public'
UMLClass='_11_0_df009b_1146146676640_543843_202'>
    <memberEnd xmi:idref='_11_0_df009b_1146146698031_74041_209'>
      <memberEnd xmi:idref='_11_0_df009b_1146146698031_363804_210'>
        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146146698031_363804_210'
name='extension$setTimer' visibility='private'
owningAssociation='_11_0_df009b_1146146698031_316325_208'
association='_11_0_df009b_1146146698031_316325_208'
type='_11_0_df009b_1146146676640_543843_202'>
          <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146698031_375714_212' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146698031_363804_210'>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146698031_935660_211' visibility='public'
owningLower='_11_0_df009b_1146146698031_363804_210'>
              </ownedEnd>
            </nestedClassifier>
          <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146698031_74041_209'
name='base$WriteVariableAction' visibility='private'
UMLClass='_11_0_df009b_1146146676640_543843_202'
association='_11_0_df009b_1146146698031_316325_208'>
            <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704909923_687782_9201'>
              <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::StructuredActions::WriteVariableAction' referentType='Class'>
                  </xmi:Extension>
                </type>
              </ownedAttribute>
            <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146742718_443918_213'
name='timer' visibility='public' UMLClass='_11_0_df009b_1146146676640_543843_202'
type='_10_5_1_df009b_1140107200765_961902_732'>
              <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146757390_827392_215' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146742718_443918_213'>
                <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146757390_862374_214' value='1' visibility='public'
owningLower='_11_0_df009b_1146146742718_443918_213'>
                  </ownedAttribute>
                <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146763062_513024_216'
name='timeout' visibility='public' UMLClass='_11_0_df009b_1146146676640_543843_202'
type='_11_0_df009b_1146146797156_830540_217'>
                  </ownedStereotype>
                <ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146146797156_830540_217' name='timeExpression'
visibility='public'>
                  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146146817000_50419_223' visibility='public'
UMLClass='_11_0_df009b_1146146797156_830540_217'>
                    <memberEnd xmi:idref='_11_0_df009b_1146146817000_890844_224'>
                      <memberEnd xmi:idref='_11_0_df009b_1146146817000_547328_225'>
                        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146146817000_547328_225'
name='extension$timeExpression' visibility='private'

```

```

owningAssociation='_11_0_df009b_1146146817000_50419_223'
association='_11_0_df009b_1146146817000_50419_223'
type='_11_0_df009b_1146146797156_830540_217'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146817000_502453_227' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146817000_547328_225'>/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146817000_888594_226' visibility='public'
owningLower='_11_0_df009b_1146146817000_547328_225'>/>
</ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146817000_890844_224'
name='base$StringExpression' visibility='private'
UMLClass='_11_0_df009b_1146146797156_830540_217'
association='_11_0_df009b_1146146817000_50419_223'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704900715_890391_8913'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::AuxiliaryConstructs::Templates::StringExpression' referentType='Class'>/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146146850593_604010_228' name='active' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146146863828_178661_234' visibility='public'
UMLClass='_11_0_df009b_1146146850593_604010_228'>
  <memberEnd xmi:idref='_11_0_df009b_1146146863828_526474_235'>/>
  <memberEnd xmi:idref='_11_0_df009b_1146146863828_694118_236'>/>
  <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146146863828_694118_236'
name='extension$active' visibility='private'
owningAssociation='_11_0_df009b_1146146863828_178661_234'
association='_11_0_df009b_1146146863828_178661_234'
type='_11_0_df009b_1146146850593_604010_228'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146863828_339463_238' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146863828_694118_236'>/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146863828_675694_237' visibility='public'
owningLower='_11_0_df009b_1146146863828_694118_236'>/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146863828_526474_235'
name='base$ReadVariableAction' visibility='private'
UMLClass='_11_0_df009b_1146146850593_604010_228'
association='_11_0_df009b_1146146863828_178661_234'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704909858_831429_9193'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::StructuredActions::ReadVariableAction' referentType='Class'>/>
  </xmi:Extension>
  </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146867031_911252_239'
name='timer' visibility='public' UMLClass='_11_0_df009b_1146146850593_604010_228'
type='_10_5_1_df009b_1140107200765_961902_732'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146880046_336496_241' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146867031_911252_239'>/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146880046_36423_240' value='1' visibility='public'
owningLower='_11_0_df009b_1146146867031_911252_239'>/>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146146892750_361720_242' name='resetTimer' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146146905859_909770_248' visibility='public'
UMLClass='_11_0_df009b_1146146892750_361720_242'>
  <memberEnd xmi:idref='_11_0_df009b_1146146905859_270653_249'>/>
  <memberEnd xmi:idref='_11_0_df009b_1146146905859_707394_250'>/>

```

```

    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146146905859_707394_250'
name='extension$resetTimer' visibility='private'
owningAssociation='_11_0_df009b_1146146905859_909770_248'
association='_11_0_df009b_1146146905859_909770_248'
type='_11_0_df009b_1146146892750_361720_242'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146905859_870274_252' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146905859_707394_250'>/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146905859_38561_251' visibility='public'
owningLower='_11_0_df009b_1146146905859_707394_250'>/>
</ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146905859_270653_249'
name='base$WriteVariableAction' visibility='private'
UMLClass='_11_0_df009b_1146146892750_361720_242'
association='_11_0_df009b_1146146905859_909770_248'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704909923_687782_9201'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::StructuredActions::WriteVariableAction' referentType='Class'>
        </xmi:Extension>
    </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146146909937_540103_253'
name='timer' visibility='public' UMLClass='_11_0_df009b_1146146892750_361720_242'
type='_10_5_1_df009b_1140107200765_961902_732'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146919062_310868_255' value='1' visibility='public'
owningUpper='_11_0_df009b_1146146909937_540103_253'>/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146146919062_158128_254' value='1' visibility='public'
owningLower='_11_0_df009b_1146146909937_540103_253'>/>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_df009b_1146151163593_904243_262' name='signalList' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146151178656_702904_268' visibility='public'
UMLClass='_11_0_df009b_1146151163593_904243_262'>
        <memberEnd xmi:idref='_11_0_df009b_1146151178656_784039_269'>/>
        <memberEnd xmi:idref='_11_0_df009b_1146151178656_348218_270'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146151178656_348218_270'
name='extension$signalList' visibility='private'
owningAssociation='_11_0_df009b_1146151178656_702904_268'
association='_11_0_df009b_1146151178656_702904_268'
type='_11_0_df009b_1146151163593_904243_262'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146151178656_636287_272' value='1' visibility='public'
owningUpper='_11_0_df009b_1146151178656_348218_270'>/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146151178656_237265_271' visibility='public'
owningLower='_11_0_df009b_1146151178656_348218_270'>/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146151178656_784039_269'
name='base$Signal' visibility='private'
UMLClass='_11_0_df009b_1146151163593_904243_262'
association='_11_0_df009b_1146151178656_702904_268'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'>
            </xmi:Extension>
        </type>
    </ownedAttribute>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146151182062_33583_273'
name='ownedSignal' visibility='public'
UMLClass='_11_0_df009b_1146151163593_904243_262'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907512_504308_9134'>

```



```

    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::Signal' referentType='Class'/>
    </xmi:Extension>
  </type>
  <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146151204562_230279_275' value='*' visibility='public'
owningUpper='_11_0_df009b_1146151182062_33583_273'/>
  <lowerValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146151204562_203975_274' value='*' visibility='public'
owningLower='_11_0_df009b_1146151182062_33583_273'/>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146831150601_751772_1'
name='Xtask' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146831187960_666594_7' visibility='public'
UMLClass='_11_0_df009b_1146831150601_751772_1'>
    <memberEnd xmi:idref='_11_0_df009b_1146831187960_737347_8'/>
    <memberEnd xmi:idref='_11_0_df009b_1146831187960_277604_9'/>
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146831187960_277604_9'
name='extension$Xtask' visibility='private'
owningAssociation='_11_0_df009b_1146831187960_666594_7'
association='_11_0_df009b_1146831187960_666594_7'
type='_11_0_df009b_1146831150601_751772_1'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146831187976_469939_11' value='1' visibility='public'
owningUpper='_11_0_df009b_1146831187960_277604_9'/>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146831187960_322353_10' visibility='public'
owningLower='_11_0_df009b_1146831187960_277604_9'/>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146831187960_737347_8'
name='base$Action' visibility='private' UMLClass='_11_0_df009b_1146831150601_751772_1'
association='_11_0_df009b_1146831187960_666594_7'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704887258_863804_8181'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Actions::BasicActions::Action' referentType='Class'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146831193929_396382_12'
name='body' visibility='public' UMLClass='_11_0_df009b_1146831150601_751772_1'>
    <type xmi:type='uml:PrimitiveType'
href='UML_Standard_Profile.xml|_9_0_2_91a0295_1110274713995_297054_0'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML Standard
Profile::datatypes::String' referentType='PrimitiveType'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype' xmi:id='_11_0_df009b_1146831384851_353902_25'
name='XsignalEvent' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_df009b_1146831399898_751832_31' visibility='public'
UMLClass='_11_0_df009b_1146831384851_353902_25'>
    <memberEnd xmi:idref='_11_0_df009b_1146831399898_176433_32'/>
    <memberEnd xmi:idref='_11_0_df009b_1146831399898_591639_33'/>
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_df009b_1146831399898_591639_33'
name='extension$XsignalEvent' visibility='private'
owningAssociation='_11_0_df009b_1146831399898_751832_31'
association='_11_0_df009b_1146831399898_751832_31'
type='_11_0_df009b_1146831384851_353902_25'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146831399898_66240_35' value='1' visibility='public'
owningUpper='_11_0_df009b_1146831399898_591639_33'/>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146831399898_535137_34' visibility='public'
owningLower='_11_0_df009b_1146831399898_591639_33'/>
    </ownedEnd>
  </nestedClassifier>

```

```

</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146831399898_176433_32'
name='base$SignalEvent' visibility='private'
UMLClass='_11_0_df009b_1146831384851_353902_25'
association='_11_0_df009b_1146831399898_751832_31'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704907577_698817_9142'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::CommonBehaviors::Communications::SignalEvent' referentType='Class'/>
    </xmi:Extension>
  </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_df009b_1146831405820_18480_36'
isOrdered='true' name='assignment' visibility='public'
UMLClass='_11_0_df009b_1146831384851_353902_25'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884574_96724_7644'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Property' referentType='Class'/>
    </xmi:Extension>
  </type>
  <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_df009b_1146831558210_466212_38' value='*' visibility='public'
owningUpper='_11_0_df009b_1146831405820_18480_36'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_df009b_1146831558210_637725_37' visibility='public'
owningLower='_11_0_df009b_1146831405820_18480_36'/>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1147098760039_790171_1' name='duration' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147102943781_138804_1' visibility='public'
UMLClass='_11_0_1_df009b_1147098760039_790171_1'>
    <memberEnd xmi:idref='_11_0_1_df009b_1147102943781_664122_2'/>
    <memberEnd xmi:idref='_11_0_1_df009b_1147102943781_661472_3'/>
    <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_1_df009b_1147102943781_661472_3'
name='extension$duration' visibility='private'
owningAssociation='_11_0_1_df009b_1147102943781_138804_1'
association='_11_0_1_df009b_1147102943781_138804_1'
type='_11_0_1_df009b_1147098760039_790171_1'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147102943781_366132_5' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147102943781_661472_3'/>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147102943781_114391_4' visibility='public'
owningLower='_11_0_1_df009b_1147102943781_661472_3'/>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_1_df009b_1147102943781_664122_2'
name='base$DataType' visibility='private'
UMLClass='_11_0_1_df009b_1147098760039_790171_1'
association='_11_0_1_df009b_1147102943781_138804_1'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'/>
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1147169333267_426050_128' name='literals' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147169347408_147228_134' visibility='public'
UMLClass='_11_0_1_df009b_1147169333267_426050_128'>
    <memberEnd xmi:idref='_11_0_1_df009b_1147169347408_122270_135'/>
    <memberEnd xmi:idref='_11_0_1_df009b_1147169347408_588150_136'/>
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147169347408_588150_136' name='extension$literals'

```

```

visibility='private' owningAssociation='_11_0_1_df009b_1147169347408_147228_134'
association='_11_0_1_df009b_1147169347408_147228_134'
type='_11_0_1_df009b_1147169333267_426050_128'>
  <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169347408_475925_138' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147169347408_588150_136'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169347408_472142_137' visibility='public'
owningLower='_11_0_1_df009b_1147169347408_588150_136'/>
</ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147169347408_122270_135' name='base$PrimitiveType'
visibility='private' UMLClass='_11_0_1_df009b_1147169333267_426050_128'
association='_11_0_1_df009b_1147169347408_147228_134'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885449_652048_7963'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::PrimitiveType' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147169353080_282614_139' name='literals' visibility='public'
UMLClass='_11_0_1_df009b_1147169333267_426050_128'>
  <type xmi:type='uml:PrimitiveType'
href='UML_Standard_Profile.xml|_9_0_2_91a0295_1110274713995_297054_0'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML Standard
Profile::datatypes::String' referentType='PrimitiveType'/>
  </xmi:Extension>
  </type>
  <upperValue xmi:type='uml:LiteralString'
xmi:id='_11_0_1_df009b_1147169376486_908720_141' value='*' visibility='public'
owningUpper='_11_0_1_df009b_1147169353080_282614_139'/>
  <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169376486_35253_140' visibility='public'
owningLower='_11_0_1_df009b_1147169353080_282614_139'/>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1147169528142_566996_142' name='struct' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147169541283_175476_148' visibility='public'
UMLClass='_11_0_1_df009b_1147169528142_566996_142'>
    <memberEnd xmi:idref='_11_0_1_df009b_1147169541283_707096_149'/>
    <memberEnd xmi:idref='_11_0_1_df009b_1147169541283_700946_150'/>
  <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147169541283_700946_150' name='extension$struct'
visibility='private' owningAssociation='_11_0_1_df009b_1147169541283_175476_148'
association='_11_0_1_df009b_1147169541283_175476_148'
type='_11_0_1_df009b_1147169528142_566996_142'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169541283_456980_152' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147169541283_700946_150'/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147169541283_475149_151' visibility='public'
owningLower='_11_0_1_df009b_1147169541283_700946_150'/>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147169541283_707096_149' name='base$DataType'
visibility='private' UMLClass='_11_0_1_df009b_1147169528142_566996_142'
association='_11_0_1_df009b_1147169541283_175476_148'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
  <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
    <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::DataType' referentType='Class'/>
  </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>

```

```

<ownedStereotype xmi:type='uml:Stereotype'
  xmi:id='_11_0_1_df009b_1147170088158_890177_172' name='real' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
    xmi:id='_11_0_1_df009b_1147170101845_606330_178' visibility='public'
    UMLClass='_11_0_1_df009b_1147170088158_890177_172'>
    <memberEnd xmi:idref='_11_0_1_df009b_1147170101845_229782_179' />
    <memberEnd xmi:idref='_11_0_1_df009b_1147170101845_482553_180' />
    <ownedEnd xmi:type='uml:ExtensionEnd'
      xmi:id='_11_0_1_df009b_1147170101845_482553_180' name='extension$real'
      visibility='private' owningAssociation='_11_0_1_df009b_1147170101845_606330_178'
      association='_11_0_1_df009b_1147170101845_606330_178'
      type='_11_0_1_df009b_1147170088158_890177_172'>
      <upperValue xmi:type='uml:LiteralInteger'
        xmi:id='_11_0_1_df009b_1147170101845_322485_182' value='1' visibility='public'
        owningUpper='_11_0_1_df009b_1147170101845_482553_180' />
      <lowerValue xmi:type='uml:LiteralInteger'
        xmi:id='_11_0_1_df009b_1147170101845_378627_181' visibility='public'
        owningLower='_11_0_1_df009b_1147170101845_482553_180' />
      </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property'
      xmi:id='_11_0_1_df009b_1147170101845_229782_179' name='base$DataType'
      visibility='private' UMLClass='_11_0_1_df009b_1147170088158_890177_172'
      association='_11_0_1_df009b_1147170101845_606330_178'>
      <type xmi:type='uml:Class'
        href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
          11.0'>
          <referenceExtension referentPath='UML Standard Profile::UML2.0
            Metamodel::Classes::Kernel::DataType' referentType='Class' />
        </xmi:Extension>
      </type>
    </ownedAttribute>
  </ownedStereotype>
  <ownedStereotype xmi:type='uml:Stereotype'
    xmi:id='_11_0_1_df009b_1147170169205_490305_187' name='time' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
      xmi:id='_11_0_1_df009b_1147170187126_796152_193' visibility='public'
      UMLClass='_11_0_1_df009b_1147170169205_490305_187'>
      <memberEnd xmi:idref='_11_0_1_df009b_1147170187126_300004_194' />
      <memberEnd xmi:idref='_11_0_1_df009b_1147170187126_344072_195' />
      <ownedEnd xmi:type='uml:ExtensionEnd'
        xmi:id='_11_0_1_df009b_1147170187126_344072_195' name='extension$time'
        visibility='private' owningAssociation='_11_0_1_df009b_1147170187126_796152_193'
        association='_11_0_1_df009b_1147170187126_796152_193'
        type='_11_0_1_df009b_1147170169205_490305_187'>
        <upperValue xmi:type='uml:LiteralInteger'
          xmi:id='_11_0_1_df009b_1147170187126_226915_197' value='1' visibility='public'
          owningUpper='_11_0_1_df009b_1147170187126_344072_195' />
        <lowerValue xmi:type='uml:LiteralInteger'
          xmi:id='_11_0_1_df009b_1147170187126_910058_196' visibility='public'
          owningLower='_11_0_1_df009b_1147170187126_344072_195' />
        </ownedEnd>
      </nestedClassifier>
      <ownedAttribute xmi:type='uml:Property'
        xmi:id='_11_0_1_df009b_1147170187126_300004_194' name='base$DataType'
        visibility='private' UMLClass='_11_0_1_df009b_1147170169205_490305_187'
        association='_11_0_1_df009b_1147170187126_796152_193'>
        <type xmi:type='uml:Class'
          href='UML_Standard_Profile.xml|_9_0_62a020a_1105704885376_903292_7939'>
          <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
            11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
              Metamodel::Classes::Kernel::DataType' referentType='Class' />
          </xmi:Extension>
        </type>
      </ownedAttribute>
    </ownedStereotype>
    <ownedStereotype xmi:type='uml:Stereotype'
      xmi:id='_11_0_1_df009b_1147170553955_357355_199' name='constant' visibility='public'>
      <nestedClassifier xmi:type='uml:Extension'
        xmi:id='_11_0_1_df009b_1147170576923_734289_205' visibility='public'
        UMLClass='_11_0_1_df009b_1147170553955_357355_199'>
        <memberEnd xmi:idref='_11_0_1_df009b_1147170576923_939012_206' />
        <memberEnd xmi:idref='_11_0_1_df009b_1147170576923_255134_207' />

```

```

    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1147170576923_255134_207' name='extension$constant'
visibility='private' owningAssociation='_11_0_1_df009b_1147170576923_734289_205'
association='_11_0_1_df009b_1147170576923_734289_205'
type='_11_0_1_df009b_1147170553955_357355_199'>
    <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147170576923_610816_209' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147170576923_255134_207'>/>
    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147170576923_271829_208' visibility='public'
owningLower='_11_0_1_df009b_1147170576923_255134_207'>/>
    </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1147170576923_939012_206' name='base$Property'
visibility='private' UMLClass='_11_0_1_df009b_1147170553955_357355_199'
association='_11_0_1_df009b_1147170576923_734289_205'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704884574_96724_7644'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Classes::Kernel::Property' referentType='Class'>/>
        </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1147346233817_396375_1' name='XmergeNode' visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1147346262270_350091_7' visibility='public'
UMLClass='_11_0_1_df009b_1147346233817_396375_1'>
        <memberEnd xmi:idref='_11_0_1_df009b_1147346262270_89764_8'>/>
        <memberEnd xmi:idref='_11_0_1_df009b_1147346262270_79029_9'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_1_df009b_1147346262270_79029_9'
name='extension$XmergeNode' visibility='private'
owningAssociation='_11_0_1_df009b_1147346262270_350091_7'
association='_11_0_1_df009b_1147346262270_350091_7'
type='_11_0_1_df009b_1147346233817_396375_1'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147346262270_138127_11' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1147346262270_79029_9'>/>
            <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1147346262270_158068_10' visibility='public'
owningLower='_11_0_1_df009b_1147346262270_79029_9'>/>
        </ownedEnd>
    </nestedClassifier>
    <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_1_df009b_1147346262270_89764_8'
name='base$DecisionNode' visibility='private'
UMLClass='_11_0_1_df009b_1147346233817_396375_1'
association='_11_0_1_df009b_1147346262270_350091_7'>
        <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704921751_161098_9437'>
            <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
                <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::IntermediateActivities::DecisionNode' referentType='Class'>/>
            </xmi:Extension>
        </type>
    </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1155205833590_482169_1' name='informationFlow'
visibility='public'>
    <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1155205850278_821704_7' visibility='public'
UMLClass='_11_0_1_df009b_1155205833590_482169_1'>
        <memberEnd xmi:idref='_11_0_1_df009b_1155205850278_375821_8'>/>
        <memberEnd xmi:idref='_11_0_1_df009b_1155205850278_776258_9'>/>
        <ownedEnd xmi:type='uml:ExtensionEnd' xmi:id='_11_0_1_df009b_1155205850278_776258_9'
name='extension$informationFlow' visibility='private'
owningAssociation='_11_0_1_df009b_1155205850278_821704_7'
association='_11_0_1_df009b_1155205850278_821704_7'
type='_11_0_1_df009b_1155205833590_482169_1'>
            <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1155205850278_937656_11' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1155205850278_776258_9'>/>

```

```

    <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1155205850278_120277_10' visibility='public'
owningLower='_11_0_1_df009b_1155205850278_776258_9'/'>
  </ownedEnd>
</nestedClassifier>
<ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_1_df009b_1155205850278_375821_8'
name='base$InformationFlow' visibility='private'
UMLClass='_11_0_1_df009b_1155205833590_482169_1'
association='_11_0_1_df009b_1155205850278_821704_7'>
  <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704899911_307257_8888'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
      <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::AuxiliaryConstructs::InformationFlows::InformationFlow'
referentType='Class'/'>
    </xmi:Extension>
  </type>
</ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1155205866044_642585_12' name='informationItem'
visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1155205883372_898466_18' visibility='public'
UMLClass='_11_0_1_df009b_1155205866044_642585_12'>
    <memberEnd xmi:idref='_11_0_1_df009b_1155205883372_83689_19'/'>
    <memberEnd xmi:idref='_11_0_1_df009b_1155205883372_721546_20'/'>
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1155205883372_721546_20' name='extension$informationItem'
visibility='private' owningAssociation='_11_0_1_df009b_1155205883372_898466_18'
association='_11_0_1_df009b_1155205883372_898466_18'
type='_11_0_1_df009b_1155205866044_642585_12'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1155205883372_252486_22' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1155205883372_721546_20'/'>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1155205883372_439354_21' visibility='public'
owningLower='_11_0_1_df009b_1155205883372_721546_20'/'>
    </ownedEnd>
  </nestedClassifier>
  <ownedAttribute xmi:type='uml:Property' xmi:id='_11_0_1_df009b_1155205883372_83689_19'
name='base$InformationItem' visibility='private'
UMLClass='_11_0_1_df009b_1155205866044_642585_12'
association='_11_0_1_df009b_1155205883372_898466_18'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704899836_442270_8880'>
      <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
        <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::AuxiliaryConstructs::InformationFlows::InformationItem'
referentType='Class'/'>
      </xmi:Extension>
    </type>
  </ownedAttribute>
</ownedStereotype>
<ownedStereotype xmi:type='uml:Stereotype'
xmi:id='_11_0_1_df009b_1155205946356_132172_23' name='if' visibility='public'>
  <nestedClassifier xmi:type='uml:Extension'
xmi:id='_11_0_1_df009b_1155205960184_458665_29' visibility='public'
UMLClass='_11_0_1_df009b_1155205946356_132172_23'>
    <memberEnd xmi:idref='_11_0_1_df009b_1155205960184_298367_30'/'>
    <memberEnd xmi:idref='_11_0_1_df009b_1155205960184_526719_31'/'>
    <ownedEnd xmi:type='uml:ExtensionEnd'
xmi:id='_11_0_1_df009b_1155205960184_526719_31' name='extension$if'
visibility='private' owningAssociation='_11_0_1_df009b_1155205960184_458665_29'
association='_11_0_1_df009b_1155205960184_458665_29'
type='_11_0_1_df009b_1155205946356_132172_23'>
      <upperValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1155205960184_713905_33' value='1' visibility='public'
owningUpper='_11_0_1_df009b_1155205960184_526719_31'/'>
      <lowerValue xmi:type='uml:LiteralInteger'
xmi:id='_11_0_1_df009b_1155205960184_577097_32' visibility='public'
owningLower='_11_0_1_df009b_1155205960184_526719_31'/'>
    </ownedEnd>
  </nestedClassifier>

```

```

    <ownedAttribute xmi:type='uml:Property'
xmi:id='_11_0_1_df009b_1155205960184_298367_30' name='base$ConditionalNode'
visibility='private' UMLClass='_11_0_1_df009b_1155205946356_132172_23'
association='_11_0_1_df009b_1155205960184_458665_29'>
    <type xmi:type='uml:Class'
href='UML_Standard_Profile.xml|_9_0_62a020a_1105704892132_915377_8442'>
        <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML
11.0'>
            <referenceExtension referentPath='UML Standard Profile::UML2.0
Metamodel::Activities::StructuredActivities::ConditionalNode' referentType='Class'/>
        </xmi:Extension>
    </type>
</ownedAttribute>
</ownedStereotype>
<ownedComment xmi:type='uml:Comment' xmi:id='_10_5_1_df009b_1141685243977_126128_690'
body='UML2 Profile for Communicating Systems'
annotatedElement='_10_5_1_df009b_1140096594546_163685_2'
owningElement='_10_5_1_df009b_1140096594546_163685_2'/>
<ownedMember xmi:type='uml:Interface' xmi:id='_11_0_1_df009b_1146146279515_868848_186'
name='Scheduler' visibility='public'
owningPackage='_10_5_1_df009b_1140096594546_163685_2'/>
<ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147169951783_357389_163'
name='PID' visibility='public' owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
        <modelExtension>
            <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
xmi:id='_11_0_1_df009b_1147169971330_732335_164' visibility='public'
classifier='_10_5_1_df009b_1140176423046_579854_35'
stereotypedElement='_11_0_1_df009b_1147169951783_357389_163'/>
        </modelExtension>
    </xmi:Extension>
</ownedMember>
<ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147169990173_61335_165'
name='Natural' visibility='public'
owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
        <modelExtension>
            <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
xmi:id='_11_0_1_df009b_1147169999173_367486_166' visibility='public'
classifier='_10_5_1_df009b_1141226913868_167015_565'
stereotypedElement='_11_0_1_df009b_1147169990173_61335_165'/>
        </modelExtension>
    </xmi:Extension>
</ownedMember>
<ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147170006673_663635_167'
name='Integer' visibility='public'
owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
        <modelExtension>
            <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
xmi:id='_11_0_1_df009b_1147170020111_613661_168' visibility='public'
classifier='_10_5_1_df009b_1141226929978_616494_571'
stereotypedElement='_11_0_1_df009b_1147170006673_663635_167'/>
        </modelExtension>
    </xmi:Extension>
</ownedMember>
<ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147170047001_504579_169'
name='Boolean' visibility='public'
owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
        <modelExtension>
            <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
xmi:id='_11_0_1_df009b_1147170054189_78262_170' visibility='public'
classifier='_11_0_1_df009b_1146146360328_775013_190'
stereotypedElement='_11_0_1_df009b_1147170047001_504579_169'/>
        </modelExtension>
    </xmi:Extension>
</ownedMember>
<ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147170070392_641428_171'
name='Real' visibility='public' owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
        <modelExtension>
            <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
xmi:id='_11_0_1_df009b_1147170114330_513889_183' visibility='public'
classifier='_11_0_1_df009b_1147170088158_890177_172'
stereotypedElement='_11_0_1_df009b_1147170070392_641428_171'/>
        </modelExtension>
    </xmi:Extension>

```

```

    </xmi:Extension>
  </ownedMember>
  <ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147170124642_292653_184'
    name='Duration' visibility='public'
    owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
      <modelExtension>
        <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
          xmi:id='_11_0_1_df009b_1147170132751_423187_185' visibility='public'
          classifier='_11_0_1_df009b_1147098760039_790171_1'
          stereotypedElement='_11_0_1_df009b_1147170124642_292653_184' />
        </modelExtension>
      </xmi:Extension>
    </ownedMember>
  <ownedMember xmi:type='uml:DataType' xmi:id='_11_0_1_df009b_1147170145017_68780_186'
    name='Time' visibility='public' owningPackage='_10_5_1_df009b_1140096594546_163685_2'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
      <modelExtension>
        <appliedStereotypeInstance xmi:type='uml:InstanceSpecification'
          xmi:id='_11_0_1_df009b_1147170198814_485055_198' visibility='public'
          classifier='_11_0_1_df009b_1147170169205_490305_187'
          stereotypedElement='_11_0_1_df009b_1147170145017_68780_186' />
        </modelExtension>
      </xmi:Extension>
    </ownedMember>
  </ownedMember>
  <ownedMember xmi:type='uml:Package'
    href='UML_Standard_Profile.xml|magicdraw_uml_standard_profile_v_0001'>
    <xmi:Extension xmi:Extender='MagicDraw UML 11.0' xmi:ExtenderID='MagicDraw UML 11.0'>
      <referenceExtension referentPath='UML Standard Profile' referentType='Package' />
    </xmi:Extension>
  </ownedMember>
</uml:Model>
</xmi:XMI>

```


Appendix B: XSLT Stylesheet for UML CS

The following XSLT stylesheet has been developed to allow a mapping from UML CS models to SDL-96. SDL-96, instead of SDL-2000, has been chosen, as there are several commercial tools available while there is currently none for SDL-2000. The XSLT stylesheet is processed by an XSL processor and it expects an XMI 2.1-based UML model file as input. Its output is an SDL-96 textual system description based on the OCL mapping rules given in Chapter 8. The XSLT stylesheet is decomposed into four separate files to improve readability. The following *umlcsmapping.xsl* stylesheet file imports the remaining stylesheets. Due some vendor-specific deviations in the XMI-implementation, some elements are not bound to their owning classifier and had to be excluded from mapping.

UMLCSMAPPING.XSL

```
<xsl:stylesheet
  version="1.0"
  xmlns:uml="http://schema.omg.org/spec/UML/2.0"
  xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

<xsl:import href="xmi.xsl"/>
<xsl:import href="system.xsl"/>
<xsl:import href="procedure.xsl"/>

<!-- show debug information? -->
<xsl:param name="debugflag" select="0"/>

<!-- strip empty lines and unnecessary whitespaces -->
<xsl:output indent="no"/>
<xsl:strip-space elements="*" />

<xsl:template match="ownedMember[@xmi:type='uml:Class']">
<!-- <xsl:for-each select="ownedMember[@xmi:type='uml:Class']"-->
  <xsl:variable name="thisid" select="@xmi:id"/>
  <xsl:variable name="profileref"
    select="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi
      :Extension/referenceExtension"/>
  <xsl:variable name="profilename" select="$profileref/@referentPath"/>
  <xsl:if test="$debugflag > 0">
    DEBUG: resolved profile name: <xsl:value-of select="$profilename"/></xsl:if>
  <xsl:if test="$profilename='UML CS Profile::system'">
    <xsl:apply-templates mode="systemgo" select="self::node()"/>
  </xsl:if>
  <xsl:if test="$profilename='Xprocess'">
    <xsl:apply-templates mode="processgo" select="self::node()"/>
  </xsl:if>
<!--/xsl:for-each-->
</xsl:template>

<xsl:template mode="goblock" match="nestedClassifier">
block type <xsl:value-of select="@name"/>;

  <xsl:for-each select="ownedConnector">
    CHANNEL
  </xsl:for-each>

<!-- define signals -->
<xsl:for-each select="nestedClassifier[@xmi:type='uml:Signal']">
signal <xsl:value-of select="@name"/>;
</xsl:for-each>
```

```

<!-- define process references -->
<xsl:for-each select="nestedClassifier[@xmi:type='uml:Class']">
  <xsl:variable name="thisid" select="@xmi:id"/>
  <xsl:variable name="profileref"
    select="xmi:Extension/modelExtension/appliedStereotypeInstance/@classifier"/>
  <xsl:variable name="profilename"
    select="//ownedMember[@xmi:type='uml:Profile']/ownedStereotype[@xmi:id=$profileref]/@name"/>

  <xsl:if test="$profilename='Xprocess'">
process type <xsl:value-of select="@name"/> referenced;
  </xsl:if>
</xsl:for-each>

<!-- defines processes -->
<xsl:for-each select="ownedAttribute[@xmi:type='uml:Property']">
  <xsl:variable name='prctype' select="@type"/>
  <xsl:variable name="prcref" select="//nestedClassifier[@xmi:id=$prctype]/@name"/>
process <xsl:value-of select="@name"/> : <xsl:value-of select="$prcref"/>;
</xsl:for-each>

endblock type <xsl:value-of select="@name"/>;
</xsl:template>

<xsl:template mode="processgo2" match="nestedClassifier">
  process <xsl:value-of select="@name"/>;

  <!-- TODO: "gate portA out with sig1; in with sig1;" -->
  <!-- FIXME: duplicate? -->
  <xsl:for-each select="ownedPort">
    <!-- owned signals from ownedPort -->
    signalroute <xsl:value-of select="@name"/>
      from <xsl:value-of select="../@name"/>
      <xsl:variable name="thisrole" select="@end"/>
      <xsl:variable name="thistype" select="@type"/>

      to <xsl:value-of
        select="//ownedMember[ownedConnector/end/@xmi:id=$thisrole]/@name"/>
    <xsl:value-of
      select="//ownedMember[nestedClassifier/ownedConnector/end/@xmi:id=$thisrole]/@name"/> with

    <xsl:variable name="thisclassifier"
      select="//nestedClassifier[@xmi:id=$thistype]"/>

    <xsl:for-each
      select="$thisclassifier/xmi:Extension/modelExtension/appliedStereotypeInstance/slot/xmi:Extension">
      <xsl:variable name="endid" select="modelExtension/value/@element"/>
      <xsl:value-of select="//nestedClassifier[@xmi:id=$endid]/@name"/>
    </xsl:for-each>
  </xsl:for-each>

  <xsl:for-each select="ownedAttribute[@xmi:type='uml:Property']">
    <xsl:variable name="thistype" select="@type"/>
    dcl <xsl:value-of select="@name"/><xsl:text> </xsl:text><xsl:choose>
      <xsl:when test="@type">
        <xsl:variable name="typeref" select="@type"/>
        <xsl:value-of select="//nestedClassifier[@xmi:id=$typeref]/@name"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="datatype"
          select="type/xmi:Extension/referenceExtension/@referentPath"/>
        <xsl:choose>
          <xsl:when test="$datatype='UML CS Profile::PID'">PID</xsl:when>
          <xsl:when test="$datatype='UML CS Profile::Natural'">Natural</xsl:when>
          <xsl:when test="$datatype='UML CS Profile::Integer'">Integer</xsl:when>

```

```

        <xsl:when test="$datatype='UML CS Profile::Boolean'">Boolean</xsl:when>
        <xsl:otherwise>ERROR: unknown datatype</xsl:otherwise>
    </xsl:choose>
</xsl:otherwise>
</xsl:choose>;
</xsl:for-each>

<!-- create procedure references -->
<xsl:for-each select="//ownedMember/region/transition/effect">
    procedure <xsl:value-of select="@name"/> referenced;
</xsl:for-each>

<!-- TODO: declarations -->
<!--xsl:apply-templates mode="declarations"/-->
<xsl:apply-templates mode="initialstate" select="//subvertex"/>
<xsl:apply-templates />
<xsl:apply-templates mode="normalize"/>

endprocess type <xsl:value-of select="../@name"/>;

<!-- create processes -->
<xsl:for-each select="//ownedMember/region/transition/effect">
    <xsl:apply-templates select="self::node()" mode="doproc"/>
</xsl:for-each>
<!--<xsl:message terminate="yes">End of parsing.</xsl:message-->
</xsl:template>

<!-- state machine environment -->
<xsl:template mode="processgo" match="ownedBehavior">
process type <xsl:value-of select="../@name"/>;

<!-- TODO: "gate portA out with sig1; in with sig1;" -->
<!-- FIXME: duplicate? -->
    <xsl:for-each select="../ownedPort">
gate <xsl:value-of select="@name"/>;
    </xsl:for-each>

    <xsl:for-each select="../ownedAttribute[@xmi:type='uml:Property']">
    <xsl:variable name="thistype" select="@type"/>
dcl
    <xsl:value-of select="@name"/><xsl:text> </xsl:text><xsl:choose>
    <xsl:when test="@type">
        <xsl:variable name="typeref" select="@type"/>
        <xsl:value-of select="//nestedClassifier[@xmi:id=$typeref]/@name"/>
    </xsl:when>
    <xsl:otherwise>
        <xsl:variable name="datatype"
select="type/xmi:Extension/referenceExtension/@referentPath"/>
        <xsl:choose>
            <xsl:when test="$datatype='UML CS Profile::PID'">Pid</xsl:when>
            <xsl:when test="$datatype='UML CS Profile::Natural'">Natural</xsl:when>
            <xsl:when test="$datatype='UML CS Profile::Integer'">Integer</xsl:when>
            <xsl:otherwise>ERROR: unknown datatype</xsl:otherwise>
        </xsl:choose>
    </xsl:otherwise>
    </xsl:choose>;
</xsl:for-each>

<!-- create procedure references -->
<xsl:for-each select="//ownedMember/region/transition/effect">
procedure <xsl:value-of select="@name"/> referenced;
</xsl:for-each>

<!-- TODO: declarations -->
<!--xsl:apply-templates mode="declarations"/-->
<xsl:apply-templates mode="initialstate" select="//ownedStereotype"/>
<xsl:apply-templates />
<xsl:apply-templates mode="normalize"/>

```

```

endprocess type <xsl:value-of select="../@name"/>;

<!-- create processes -->
<xsl:for-each select="//ownedMember/region/transition/effect">
  <xsl:apply-templates select="self::node()" mode="doproc"/>
</xsl:for-each>
<xsl:message terminate="yes">End of parsing.</xsl:message>
</xsl:template>

<xsl:template mode="initialstate" match="//subvertex[@kind='initial']">
  <xsl:variable name="initialid" select="@xmi:id"/>
  <xsl:choose>
    <xsl:when
      test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension[@referentPath='UML CS Profile::start']">
start;
      <!-- determine the transition from the start element -->
      <!-- we don't need to do this anymore (06-06-13) -->
      <xsl:variable name="firstttrans"
        select="//appliedStereotypeInstance[@classifier=$initialid]/@xmi:id"/>
      <xsl:variable name="stateid"
        select="//subvertex[xmi:Extension/modelExtension/appliedStereotypeInstance/@xmi:id=$firstttrans]/@xmi:id"/>
      <xsl:apply-templates mode="fromSubvertex"
        select="//transition[@source=$initialid]"/>

    </xsl:when>
    <xsl:otherwise>
      <xsl:message terminate="yes">ERROR: no initial state found!</xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="*" mode="copy-without-namespaces" priority="-1">
  <xsl:element name="{name()}">
    <xsl:copy-of select="@*" />
    <xsl:apply-templates mode="copy-without-namespaces" />
  </xsl:element>
</xsl:template>

<xsl:template match="subvertex[@xmi:type='uml:State']">
  <!-- don't output history state -->
  <xsl:choose>
    <xsl:when test="@name='-'"> </xsl:when>
    <xsl:otherwise>
state <xsl:value-of select="@name"/>;
      <!-- resolve transition matching the state -->
      <xsl:variable name="stateid" select="@xmi:id"/>
      <xsl:apply-templates mode="fromSubvertex"
        select="//transition[@source=$stateid]"/>
      <!-- TODO: deferrable trigger -> save -->
endstate <xsl:value-of select="@name"/>;
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="transition" mode="fromSubvertex">
  <xsl:param name="ischoice"/>
  <!-- deal with gates, triggers, conditions here! -->
  <xsl:if test="$debugflag > 0">
DEBUG: matching transition</xsl:if>
  <!-- trigger? -->
  <xsl:if test="trigger">
    <xsl:variable name="triggerid" select="trigger/@event"/>
    <xsl:apply-templates mode="rslvtrigger"
      select="//ownedMember[@xmi:id=$triggerid]"/>
  </xsl:if>

```

```

<xsl:choose>
  <xsl:when test="$ischoice='yes'"><!-- nothing to do here --></xsl:when>
  <xsl:otherwise>
    <xsl:if test="guard">
provided <xsl:value-of select="guard/specification/@body"/>;
    </xsl:if>
  </xsl:otherwise>
</xsl:choose>
<xsl:if test="effect[@xmi:type='uml:Activity']">
<xsl:if test="$debugflag &gt; 0">
DEBUG: activity '<xsl:value-of select="effect/@name"/>' found</xsl:if>

  <xsl:for-each select="effect/node">
    <xsl:choose>
      <xsl:when
        test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::begin'">
        task {
          </xsl:when>
        <xsl:when
          test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::Xtask'">
          <xsl:value-of
            select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/value/@value"/>
          </xsl:when>
          <xsl:when test="@xmi:type='uml:ActivityFinalNode'">
            };
          </xsl:when>
        <xsl:when
          test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::output'">
          <xsl:variable name="outputid" select="@signal"/>
          output <xsl:value-of
            select="//nestedClassifier[@xmi:id=$outputid]/@name"/>
          <xsl:if
            test="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/definingFeature/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::output::attributes'">
            <xsl:value-of
              select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/value/@value"/>
            </xsl:if>
          <xsl:if
            test="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/definingFeature/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::output::via'">
            <xsl:variable name="viaid"
              select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/value/@element"/>
            via <xsl:value-of select="//ownedPort[@xmi:id=$viaid]/@name"/>;
            </xsl:if>;
          </xsl:when>
        </xsl:choose>
      </xsl:for-each>
<!--call <xsl:value-of select="effect/@name"/>;-->
    <!-- this is where we need to call the procedure... -->
  </xsl:if>
  <xsl:if test="effect[@xmi:type='uml:StateMachine']">
    effect
  </xsl:if>
  <!-- resolve transition target, call template to insert name -->
  <xsl:variable name="transid" select="@target"/>
  <xsl:apply-templates mode="fromTransition"
    select="//subvertex[@xmi:id=$transid]"/>
  <!--xsl:apply-templates/-->
</xsl:template>

```

```

<xsl:template match="transition" mode="fromSubvertex_fix">
  <xsl:param name="ischoice"/>
  <!-- deal with gates, triggers, conditions here! -->
  <xsl:if test="$debugflag &gt; 0">
    DEBUG: matching transition</xsl:if>
  <!-- trigger? -->
  <xsl:if test="trigger">
    <xsl:variable name="triggerid" select="trigger/@event"/>
    <xsl:apply-templates mode="rslvtrigger"
      select="//ownedMember[@xmi:id=$triggerid]"/>
  </xsl:if>
  <xsl:choose>
    <xsl:when test="$ischoice='yes'"><!-- nothing to do here --></xsl:when>
    <xsl:otherwise>
      <xsl:if test="guard">
        provided <xsl:value-of select="guard/specification/@body"/>;
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:if test="effect[@xmi:type='uml:Activity']">
    <xsl:if test="$debugflag &gt; 0">
      DEBUG: activity '<xsl:value-of select="effect/@name"/>' found</xsl:if>

    <xsl:for-each select="effect/node">
      <xsl:choose>
        <xsl:when
          test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::begin'">
          task {
            </xsl:when>
          <xsl:when
            test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::Xtask'">
            <xsl:value-of
              select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/value/@value"/>
            </xsl:when>
            <xsl:when test="@xmi:type='uml:ActivityFinalNode'">
              };
            </xsl:when>
          <xsl:when
            test="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::output'">
            <xsl:variable name="outputid" select="@signal"/>
            output <xsl:value-of
              select="//nestedClassifier[@xmi:id=$outputid]/@name"/>
            <xsl:if
              test="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/definingFeature/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::output::attributes'">
              <xsl:value-of
                select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/value/@value"/>
            </xsl:if>
            <xsl:if
              test="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/definingFeature/xmi:Extension/referenceExtension/@referentPath='UML CS Profile::output::via'">
              <xsl:variable name="viaid"
                select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/value/@element"/>
              via <xsl:value-of select="//ownedPort [@xmi:id=$viaid]/@name"/>;
            </xsl:if>;
            </xsl:when>
          </xsl:choose>
        </xsl:for-each>
        <!--call <xsl:value-of select="effect/@name"/>;-->
        <!-- this is where we need to call the procedure... -->

```

```

</xsl:if>
<xsl:if test="effect[@xmi:type='uml:StateMachine']">
  <!-- TODO: recursive call? -->
  effect...
</xsl:if>
<!-- resolve transition target, call template to insert name -->
<xsl:variable name="transid" select="@target"/>
<!--xsl:apply-templates mode="fromTransition"
  select="//subvertex[@xmi:id=$transid]"/-->
<!--xsl:apply-templates/-->
</xsl:template>

<xsl:template match="ownedMember" mode="rslvtrigger">
  <xsl:if test="$debugflag &gt; 0">
    DEBUG: getting trigger
  </xsl:if>
  <xsl:variable name="prop" select="@signal"/>

  <!-- handle 'TimeEvent' -->
  <xsl:if test="@xmi:type='uml:TimeEvent'">
    set(<xsl:value-of select="when/@body"/>);
  </xsl:if>

  <xsl:for-each select="ownedMember">
    input <xsl:value-of select="//nestedClassifier[@xmi:id=$prop]/@name"/>;
    <!-- FIX 06-04-12 input name resolution changed (again 06-04-21)
    input <xsl:value-of select="//ownedMember[@xmi:id=$prop]/@name"/>;-->
  </xsl:for-each>
</xsl:template>

<xsl:template match="subvertex" mode="fromTransition">
  <xsl:if test="$debugflag &gt; 0">
    DEBUG: matching subvertex</xsl:if>
  <!-- insert nextstate <name of transition target> -->
  <xsl:choose>
    <xsl:when test="@xmi:type='uml:State'">
      nextstate <xsl:value-of select="@name"/>;

    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="@kind='choice'"> <!-- pseudostate 'choice' following -->
          <xsl:if test="$debugflag &gt; 0">
            DEBUG: choice found</xsl:if>
          <xsl:variable name="foutid"><xsl:value-of
            select="outgoing/@xmi:idref"/></xsl:variable>
          <xsl:apply-templates mode="firstchoice"
            select="//transition[@xmi:id=$foutid]"/>
          <xsl:for-each select="outgoing">
            <xsl:variable name="outid"><xsl:value-of
              select="@xmi:idref"/></xsl:variable>
            <!-- run loop for every transition -->
            <xsl:apply-templates mode="fromChoice"
              select="//transition[@xmi:id=$outid]"/>
          </xsl:for-each>
        </xsl:when>
        <xsl:when test="@kind='terminate'">
          stop;
        </xsl:when>
        <xsl:when test="@kind='fork'">
          fork?!
          <xsl:variable name="outid"><xsl:value-of
            select="@xmi:idref"/></xsl:variable>
          <xsl:apply-templates mode="fromChoice"
            select="//transition[@xmi:id=$outid]"/>
        </xsl:when>
        <xsl:when test="@kind='junction'">

```

```

        <!-- TODO: junction -->
    </xsl:when>
    <xsl:otherwise>
        ERROR: undefined next state
    </xsl:otherwise>
</xsl:choose>
</xsl:otherwise>
</xsl:choose>

<xsl:for-each select="outgoing">
    <xsl:variable name="outid" select="@xmi:idref"/>
    <xsl:apply-templates mode="fromSubvertex_fix"
        select="//transition[@xmi:id=$outid]"/>
</xsl:for-each>
</xsl:template>

<xsl:template mode="firstchoice" match="transition">
    <xsl:variable name="prop" select="guard/specification/@body"/>
    <xsl:choose>
        <xsl:when test="substring-before($prop, '!=') ">
decision <xsl:value-of select="substring-before($prop, '!=') "/>;
        </xsl:when>
        <xsl:when test="substring-before($prop, '&gt;=') ">
decision <xsl:value-of select="substring-before($prop, '&gt;=') "/>;
        </xsl:when>
        <xsl:when test="substring-before($prop, '&lt;=') ">
decision <xsl:value-of select="substring-before($prop, '&lt;=') "/>;
        </xsl:when>
        <xsl:when test="substring-before($prop, '=') ">
decision <xsl:value-of select="substring-before($prop, '=') "/>;
        </xsl:when>
        <xsl:when test="substring-before($prop, '&gt;;') ">
decision <xsl:value-of select="substring-before($prop, '&gt;;') "/>;
        </xsl:when>
        <xsl:when test="substring-before($prop, '&lt;;') ">
decision <xsl:value-of select="substring-before($prop, '&lt;;') "/>;
        </xsl:when>
        <xsl:otherwise>
            ERROR: unknown choice found!
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match="ownedStereotype" mode="fromTransition">
    <!-- insert nextstate <name of transition target> -->nextstate <xsl:value-of
        select="@name"/>;
</xsl:template>

<xsl:template match="transition" mode="fromChoice">
    <xsl:param name="loopcnt" />
    <xsl:if test="$debugflag &gt; 0">
DEBUG: getting condition of choice number <xsl:value-of select="$loopcnt"/>
    </xsl:if>
    <xsl:choose>
        <xsl:when test="guard/specification/@body='true'">
true:</xsl:when>
        <xsl:when test="guard/specification/@body='false'">
false:</xsl:when>
        <xsl:when test="guard/specification/@body='else'">
else:</xsl:when>
        <xsl:otherwise> <!-- get substring of choice -->
            <xsl:variable name="prop" select="guard/specification/@body"/>
            <xsl:choose>
                <xsl:when test="substring-before($prop, '!=') ">
(=<xsl:value-of select="substring-after($prop, '!=') "/>): <!-- this should be /= ?!
                -->
            </xsl:when>

```



```

    <xsl:when test="substring-before($prop, '&gt;=') ">
(&gt;=<xsl:value-of select="substring-after($prop, '&gt;=') "/>):
    </xsl:when>
    <xsl:when test="substring-before($prop, '&lt;=') ">
(&lt;=<xsl:value-of select="substring-after($prop, '&lt;=') "/>):
    </xsl:when>
    <xsl:when test="substring-before($prop, '=') ">
(=<xsl:value-of select="substring-after($prop, '=') "/>):
    </xsl:when>
    <xsl:when test="substring-before($prop, '&gt;') ">
(&gt;<xsl:value-of select="substring-after($prop, '&gt;') "/>):
    </xsl:when>
    <xsl:when test="substring-before($prop, '&lt;') ">
(&lt;<xsl:value-of select="substring-after($prop, '&lt;') "/>):
    </xsl:when>
    <xsl:otherwise>
        ERROR: unknown choice found! <!-- terminate! -->
    </xsl:otherwise>
</xsl:choose>

</xsl:otherwise>
</xsl:choose>

<xsl:apply-templates mode="fromSubvertex" select="self::node()" >
    <xsl:with-param name="ischoice">yes</xsl:with-param>
</xsl:apply-templates>

<!--xsl:variable name="transid" select="@target"/>
<xsl:apply-templates mode="fromTransition"
    select="//subvertex[@xmi:id=$transid]"/-->
</xsl:template>

<xsl:template match="subvertex[@kind='terminate']" mode="termination">
stop;
    <xsl:apply-templates mode="normalize"/>
    <xsl:message terminate="yes">EOF</xsl:message>
</xsl:template>

<xsl:template match="text()" mode="normalize">
    <xsl:value-of select="normalize-space(.)"/>
</xsl:template>

</xsl:stylesheet>

```

PROCEDURE.XSL

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    version="1.0"
    xmlns:uml="http://schema.omg.org/spec/UML/2.0"
    xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    >

    <!-- this template creates procedures -->
    <xsl:template match="effect" mode="doproc">
procedure <xsl:value-of select="@name"/>;

        <!-- find initial node of the procedure, represented by applied 'begin'-
stereotype -->
        <xsl:variable name="nodebegin"
            select="node[xmi:Extension/modelExtension/appliedStereotypeInstance/@classifi
er="//ownedMember[@xmi:type='uml:Profile']/ownedStereotype[@name='begin']/@xmi
:id]"/>

        <!-- did we find the initial node? -->

```

```

    <xsl:choose>
      <xsl:when test="$nodebegin">
start;
        <!-- the applied template will select the target from the initial
node -->
        <xsl:apply-templates select="edge[@source=$nodebegin/@xmi:id]"
mode="doedge"/>
        </xsl:when>
        <xsl:otherwise>
          ERROR: initial node of procedure not present!
        </xsl:otherwise>
      </xsl:choose>
endprocedure;
    </xsl:template>

    <xsl:template match="edge" mode="doedge">
    <xsl:variable name="targetnode" select="@target"/>
    <xsl:apply-templates select="../node[@xmi:id=$targetnode]" mode="donode"/>
    </xsl:template>

    <!-- this template parses nodes; when available, the outgoing control flow will
be followed -->
    <xsl:template match="node" mode="donode">

      <xsl:variable name="appst"
select="xmi:Extension/modelExtension/appliedStereotypeInstance/@classifier"/>
      <xsl:variable name="sttype"
select="//ownedMember[@xmi:type='uml:Profile']/ownedStereotype[@xmi:id=$appst
]/@name"/>

      <xsl:choose>
        <xsl:when test="$sttype='input'">
input <xsl:value-of select="@name"/>;
        </xsl:when>
        <xsl:when test="$sttype='output'">
output <xsl:value-of select="@name"/>;
        </xsl:when>
        <xsl:when test="$sttype='return'">
return;
        </xsl:when>
        <xsl:otherwise>
          ERROR: unknown activity node!
        </xsl:otherwise>
      </xsl:choose>

      <xsl:if test="@outgoing">
        <!-- check if this is no return node -> outgoing not allowed! -->
        <xsl:if test="$sttype='return'">
          ERROR: outgoing attribute on return node!
        </xsl:if>
        <xsl:variable name="out" select="@xmi:id"/>
        <xsl:apply-templates select="../edge[@source=$out]" mode="doedge"/>
      </xsl:if>
    </xsl:template>
  </xsl:stylesheet>

```

SYSTEM.XSL

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:uml="http://schema.omg.org/spec/UML/2.0"
  xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

```

```

<!-- system declaration -->
<xsl:template mode="systemgo" match="ownedMember">
  system <xsl:value-of select="@name"/>;
  <!-- get nested block name -->

  <xsl:for-each select="nestedClassifier">
    <xsl:variable name="nCtype"
select="xmi:Extension/modelExtension/appliedStereotypeInstance/classifier/xmi:Extension/referenceExtension/@referentPath"/>
    <xsl:choose>
      <xsl:when test="$nCtype='UML CS Profile::enumeration'">
        newtype <xsl:value-of select="@name"/>
        literals <xsl:for-each select="ownedLiteral"><xsl:value-of
select="@name"/><xsl:if test="position() != last()">, </xsl:if> </xsl:for-each>;
        endnewtype;
      </xsl:when>
      <xsl:when test="$nCtype='UML CS Profile::struct'">
        newtype <xsl:value-of select="@name"/> struct
        <xsl:for-each select="ownedAttribute">
          <xsl:value-of select="@name"/>
          <xsl:choose>
            <xsl:when test="@type">
              <xsl:variable name="typeref" select="@type"/>
              <xsl:text> </xsl:text><xsl:value-of
select="//nestedClassifier[@xmi:id=$typeref]/@name"/>;
            </xsl:when>
            <xsl:otherwise>
              <xsl:text> </xsl:text><xsl:variable name="datatype"
select="type/xmi:Extension/referenceExtension/@referentPath"/>
            <xsl:choose>
              <xsl:when test="$datatype='UML CS
Profile::PID'">PID;</xsl:when>
              <xsl:when test="$datatype='UML CS
Profile::Natural'">Natural;</xsl:when>
              <xsl:when test="$datatype='UML CS
Profile::Integer'">Integer;</xsl:when>
              <xsl:otherwise>ERROR: unknown
datatype</xsl:otherwise>
            </xsl:choose>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:text>
        </xsl:text>
      </xsl:for-each>
      endnewtype;
    </xsl:when>
    <xsl:when test="$nCtype='UML CS Profile::signal'">
      signal <xsl:value-of select="@name"/><xsl:if
test="ownedAttribute"><xsl:for-each select="ownedAttribute">
        <xsl:choose>
          <xsl:when test="@type">
            <xsl:variable name="typeref" select="@type"/>
            <xsl:value-of
select="//nestedClassifier[@xmi:id=$typeref]/@name"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:variable name="datatype"
select="type/xmi:Extension/referenceExtension/@referentPath"/>
            <xsl:choose>
              <xsl:when test="$datatype='UML CS
Profile::PID'">PID</xsl:when>
              <xsl:when test="$datatype='UML CS
Profile::Natural'">Natural</xsl:when>
              <xsl:when test="$datatype='UML CS
Profile::Integer'">Integer</xsl:when>
              <xsl:otherwise>ERROR: unknown
datatype</xsl:otherwise>
            </xsl:choose>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </xsl:when>
  </xsl:for-each>

```

```

        </xsl:choose>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:if test="position() != last()">, </xsl:if></xsl:for-
each></xsl:if>;
    </xsl:when>
    <xsl:when test="$nCtype='UML CS Profile::block'">
        block <xsl:value-of select="@name"/> referenced;
    </xsl:when>
    <xsl:when test="$nCtype='UML CS Profile::Xprocess'">
        <!-- must be included within a block -->
        process <xsl:value-of select="@name"/> referenced;
    </xsl:when>
    <xsl:when test="$nCtype='UML CS Profile::interface'">
        (interface <xsl:value-of select="@name"/>;)
    </xsl:when>
    <xsl:when test="$nCtype='UML CS Profile::signalList'">
        signallist <xsl:value-of select="@name"/> = <xsl:for-each
select="xmi:Extension/modelExtension/appliedStereotypeInstance/slot/xmi:Extension"
select="modelExtension/value/@element"/>
        <xsl:variable name="listid"
select="//nestedClassifier[@xmi:id=$listid]/@name"/>
        <xsl:if test="position() != last()">, </xsl:if>
        </xsl:for-each>;
    </xsl:when>
    <xsl:otherwise>ERROR: unknown nested classifier
found!</xsl:otherwise>
    </xsl:choose>

</xsl:for-each>

<xsl:for-each select="//ownedConnector">
    channel
    <xsl:for-each select="end">
        <xsl:variable name="chend" select="@role"/>
        <xsl:variable name="chid"
select="//ownedPort [@xmi:id=$chend]/@type"/>
        channel <xsl:value-of select="//ownedPort [@xmi:id=$chend]/@name"/>
with (<xsl:value-of
select="//nestedClassifier[@xmi:id=$chid]/@name"/>);
        <!--xsl:if test="position()=1">from <xsl:value-of
select="//ownedPort [@xmi:id=$chend]/@name"/></xsl:if>
        <xsl:if test="position()=last()"> via <xsl:value-of
select="//ownedPort [@xmi:id=$chend]/@name"/></xsl:if-->
        </xsl:for-each>
    endchannel;
</xsl:for-each>

<xsl:for-each select="nestedClassifier[@xmi:type='uml:Class']">
    <xsl:apply-templates mode="processgo2" select="self::node()"/>
</xsl:for-each>

<!--block type <xsl:value-of select="$blockname"/> referenced;

    block randomBl : <xsl:value-of select="$blockname"/>;-->
endsystem <xsl:value-of select="@name"/>;
<!--xsl:apply-templates mode="goblock" select="nestedClassifier/-->
<xsl:apply-templates mode="processgo"
select="nestedClassifier/nestedClassifier[@xmi:type='uml:Class']/ownedBehavior[@xmi
:type='uml:StateMachine']"/>
    </xsl:template>
</xsl:stylesheet>

```

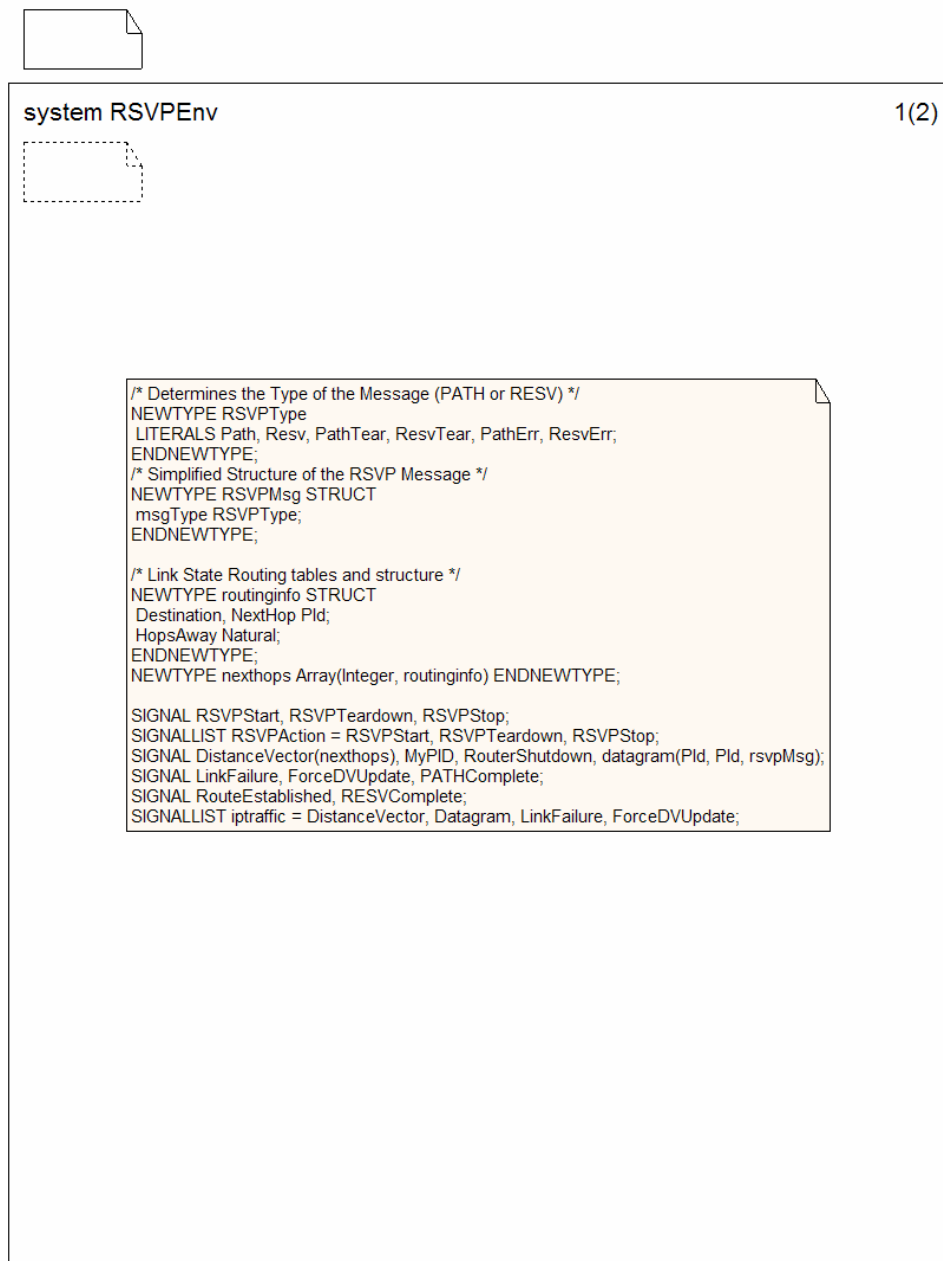
XMI.XSL

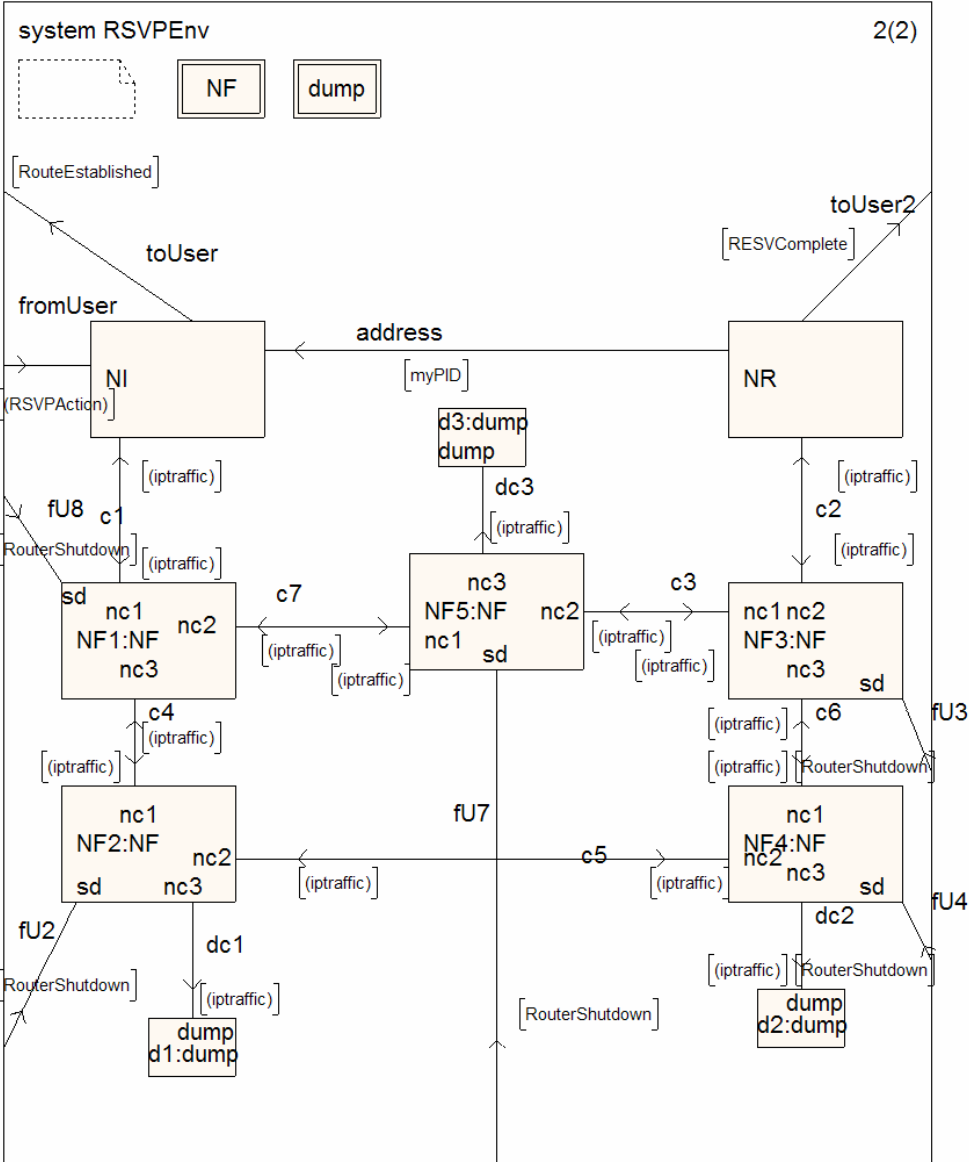
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:uml="http://schema.omg.org/spec/UML/2.0"
  xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  >

  <xsl:template match="xmi:XMI">
    <xsl:choose>
      <xsl:when test="@xmi:version='2.1'">
        <xsl:apply-templates />
      </xsl:when>
      <xsl:otherwise>
        <xsl:message terminate="yes">ERROR: Invalid XMI
version!</xsl:message>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

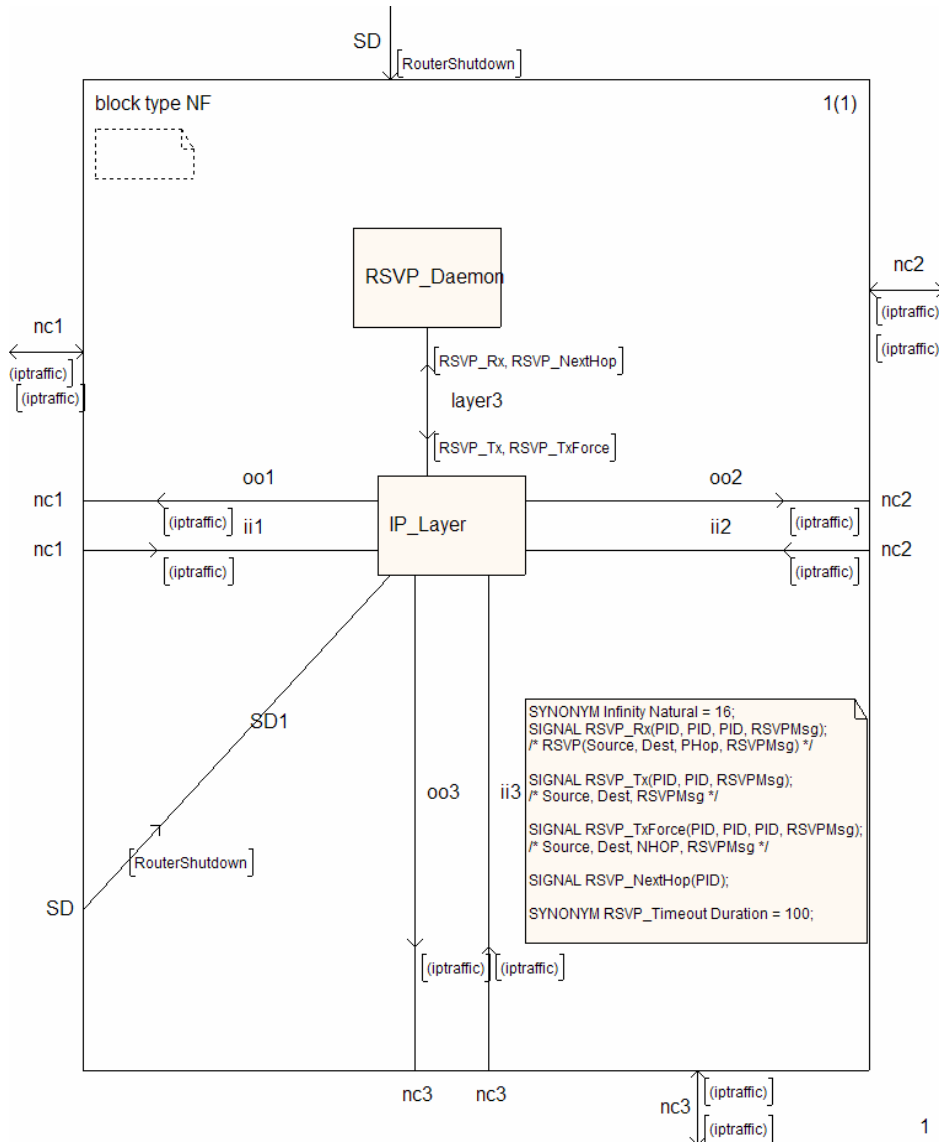

Appendix C: SDL Diagrams of the RSVP model

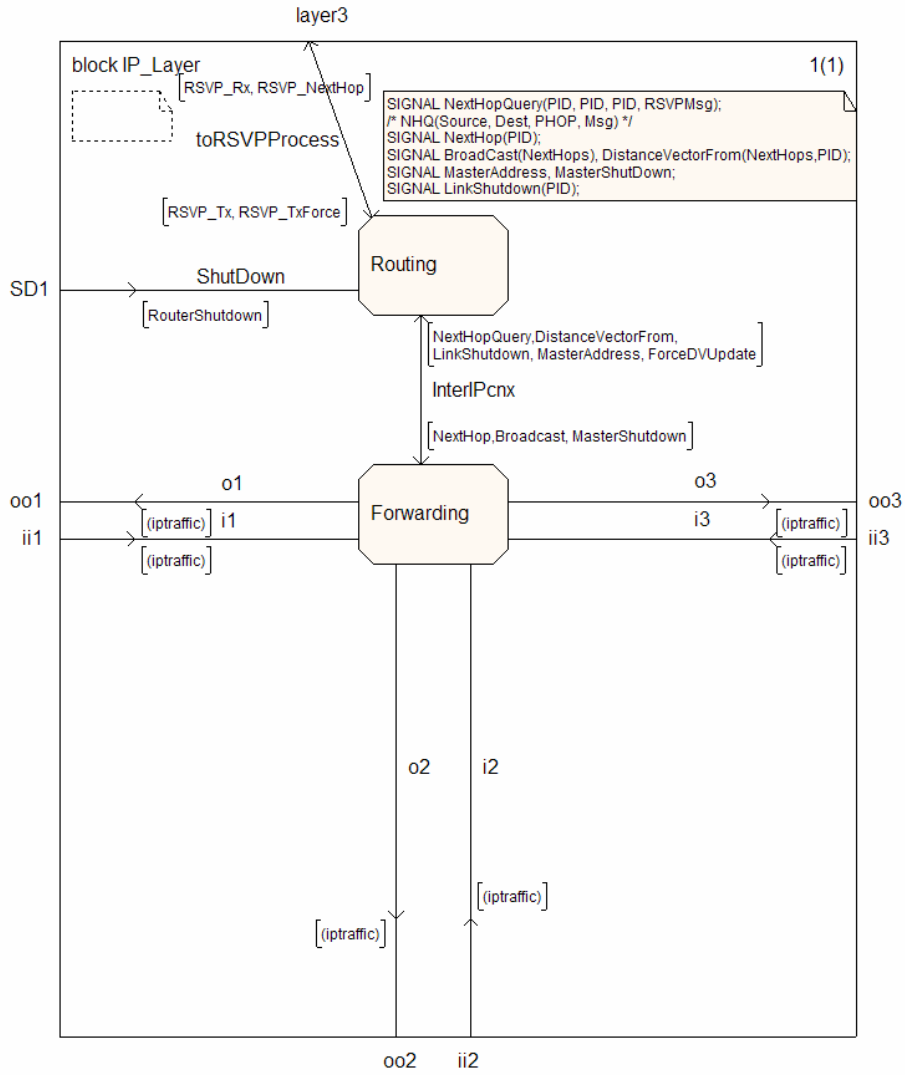
This appendix depicts all SDL block and process diagrams of the RSVP model described and analyzed in Section 5.1. This SDL model has been partly published in [WFH05]. All following SDL diagrams have been created using Tau 4.6.

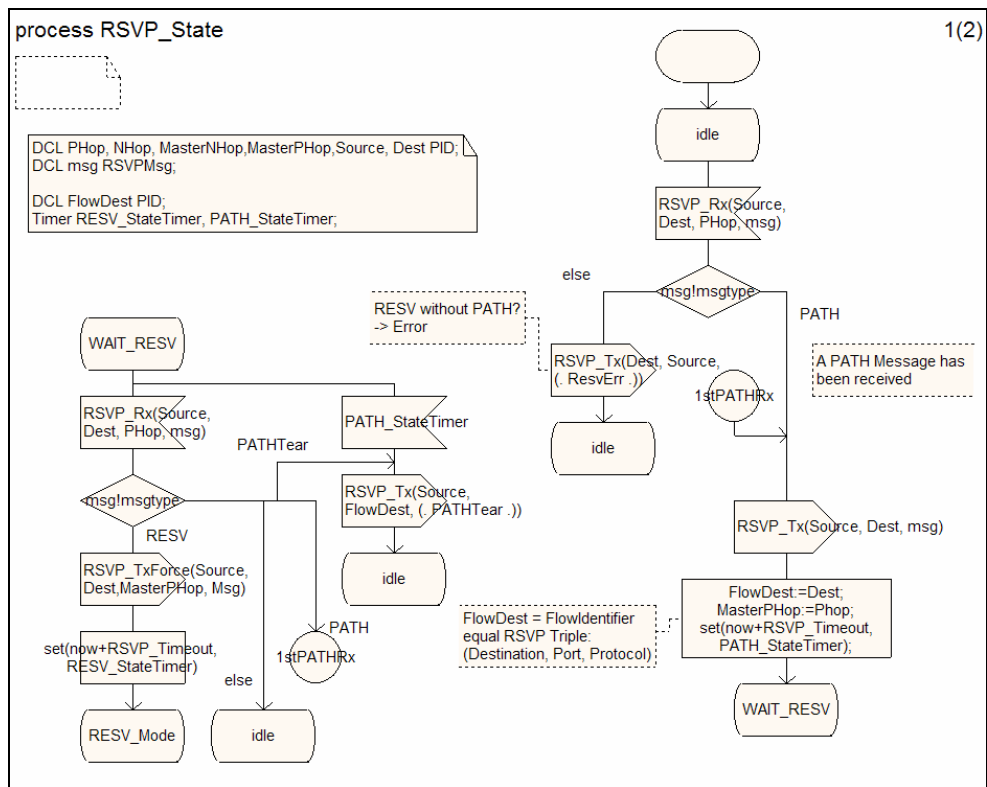
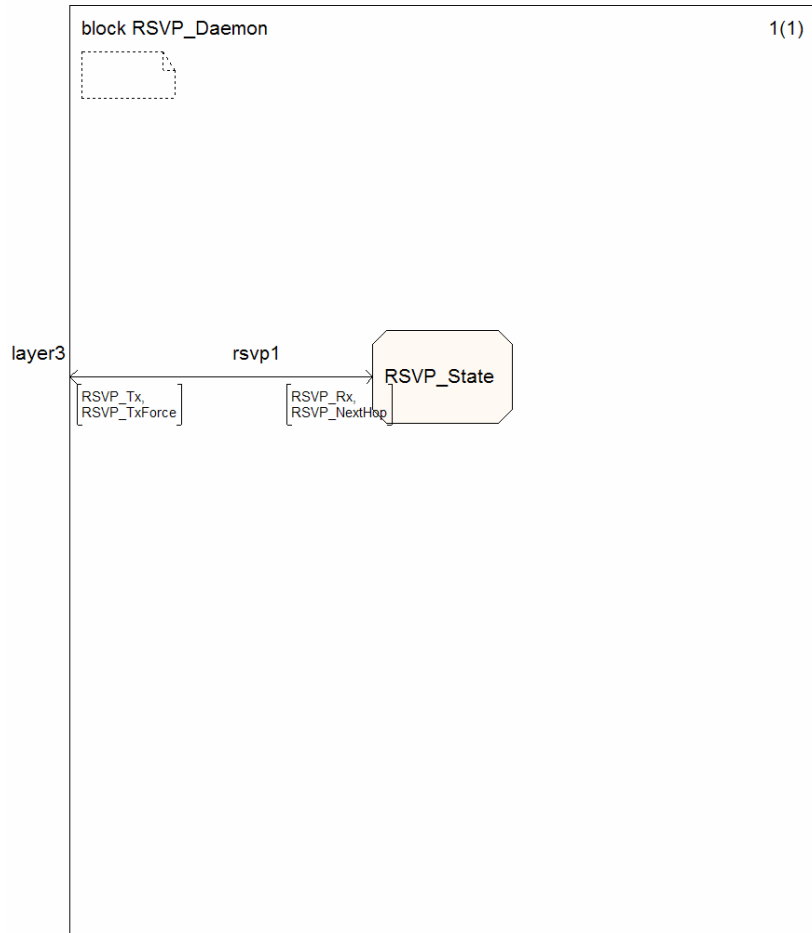


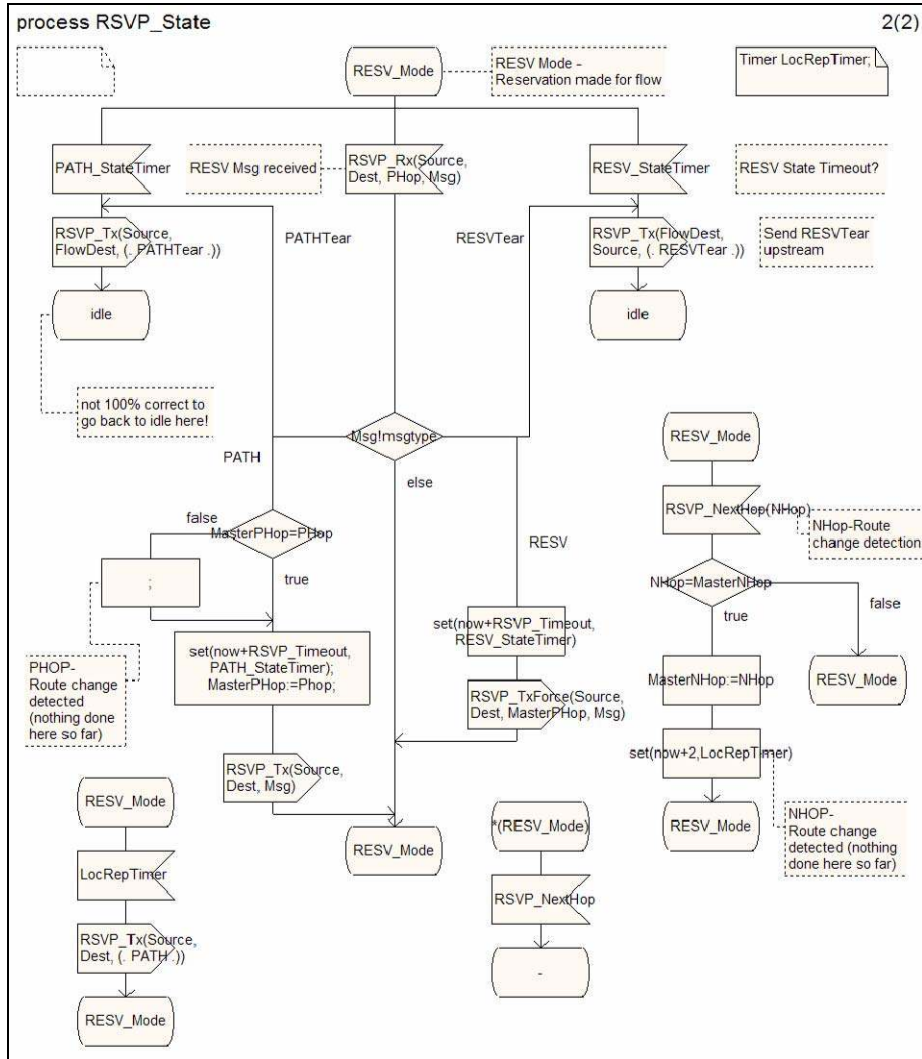


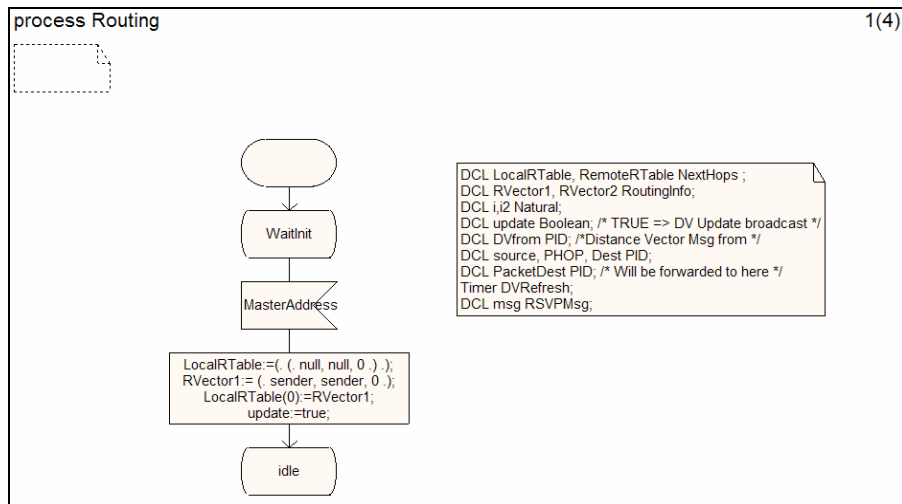
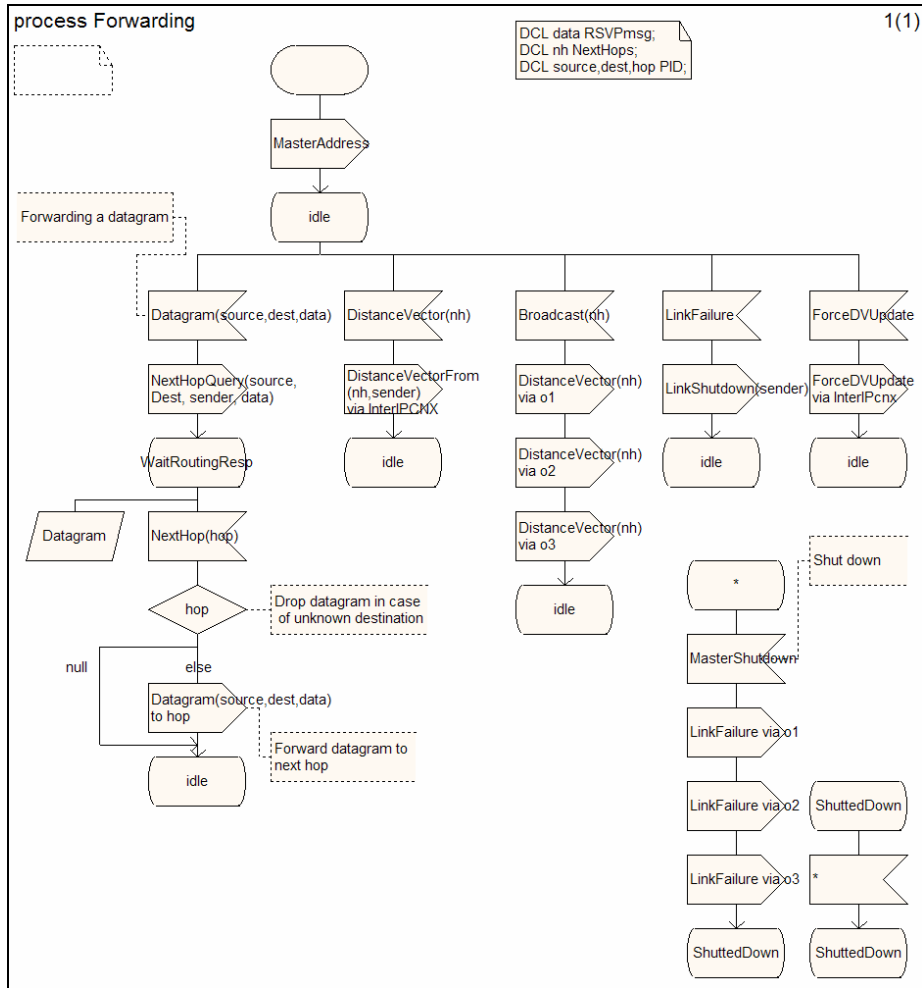
NF BLOCK AND PROCESS DIAGRAMS

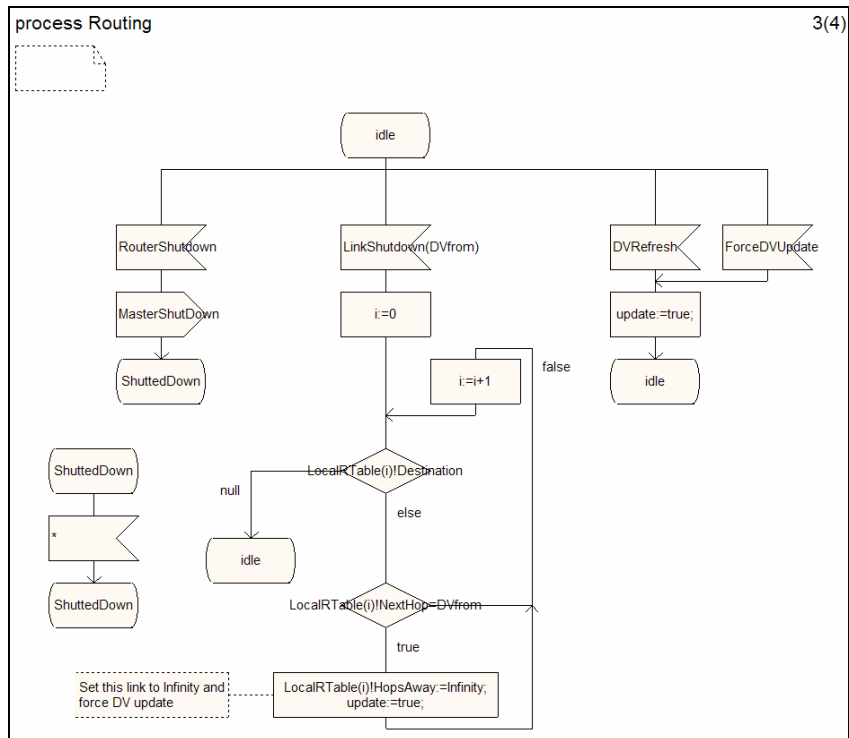
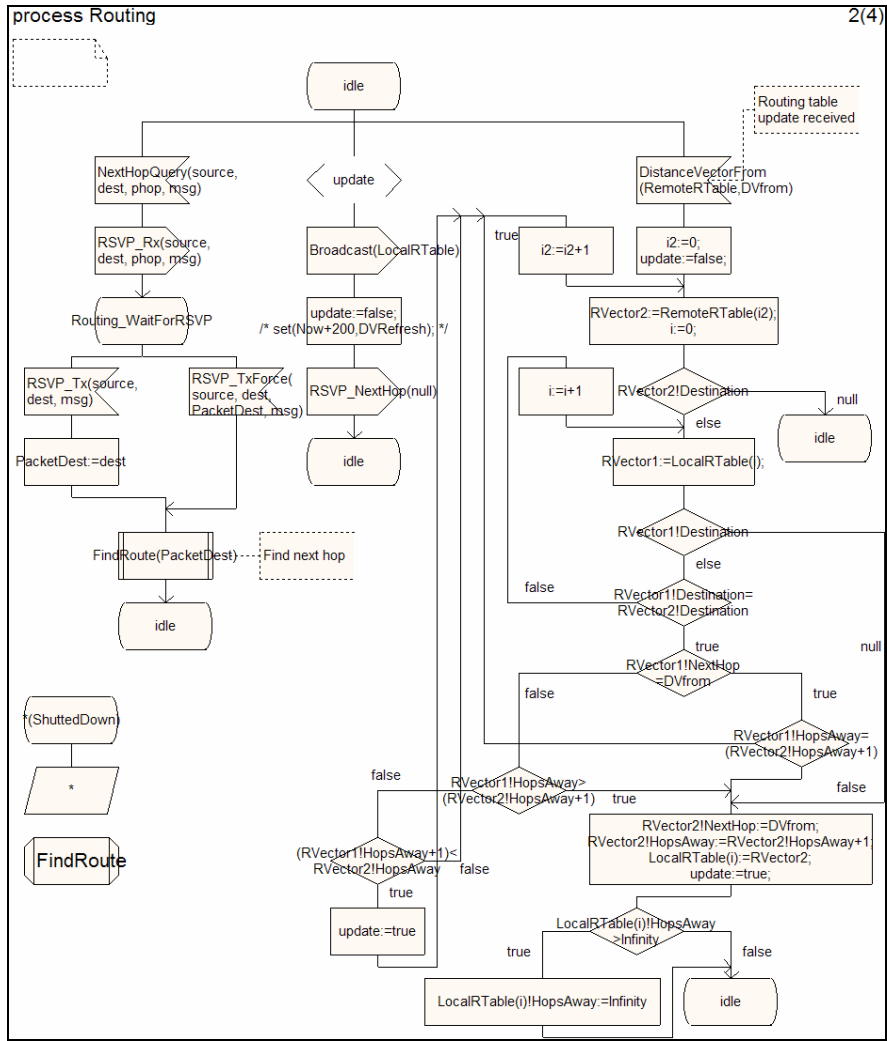




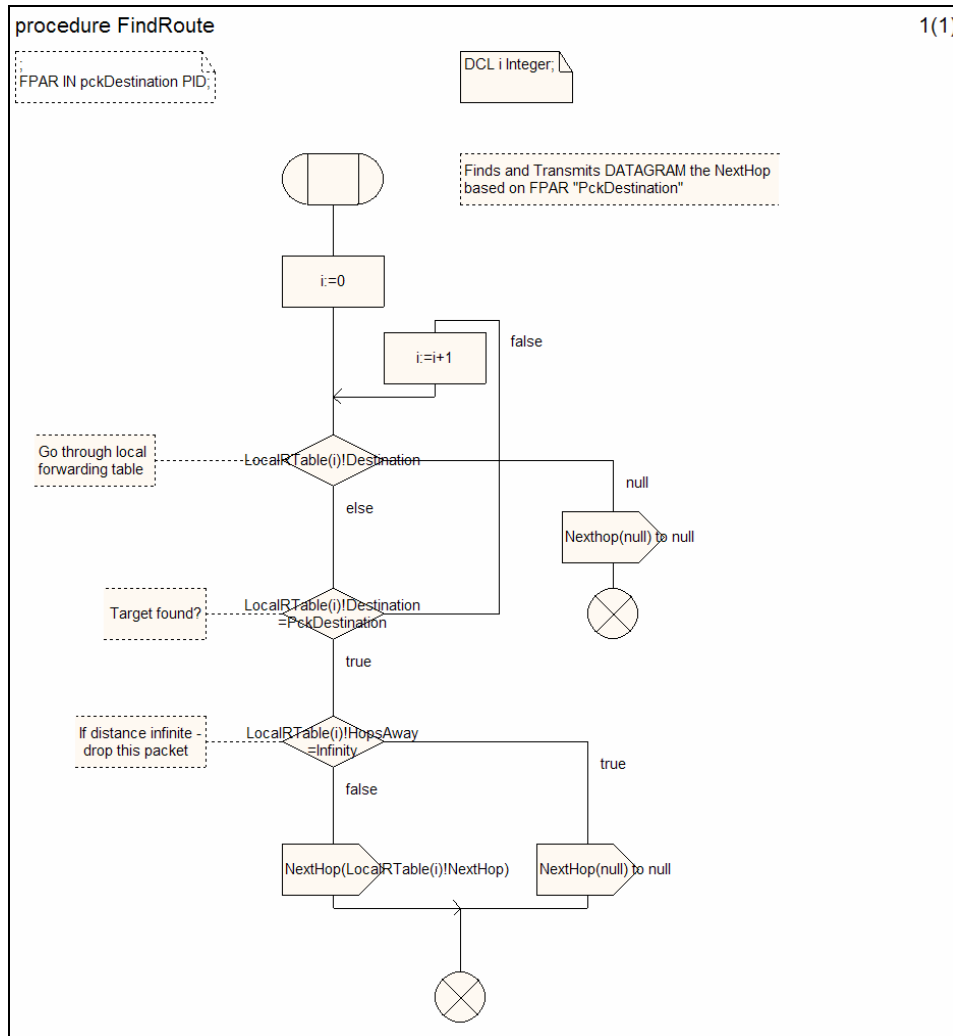




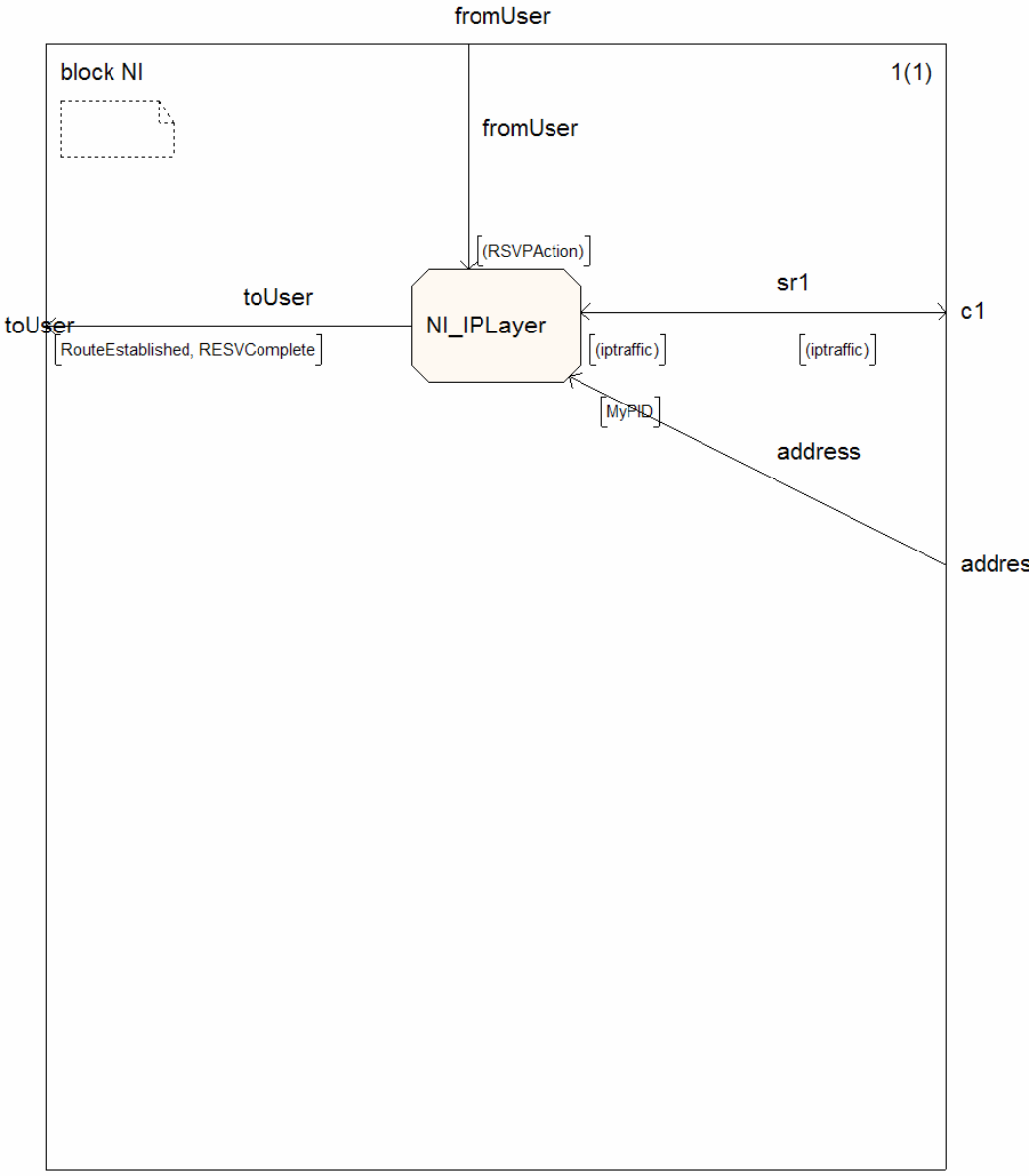


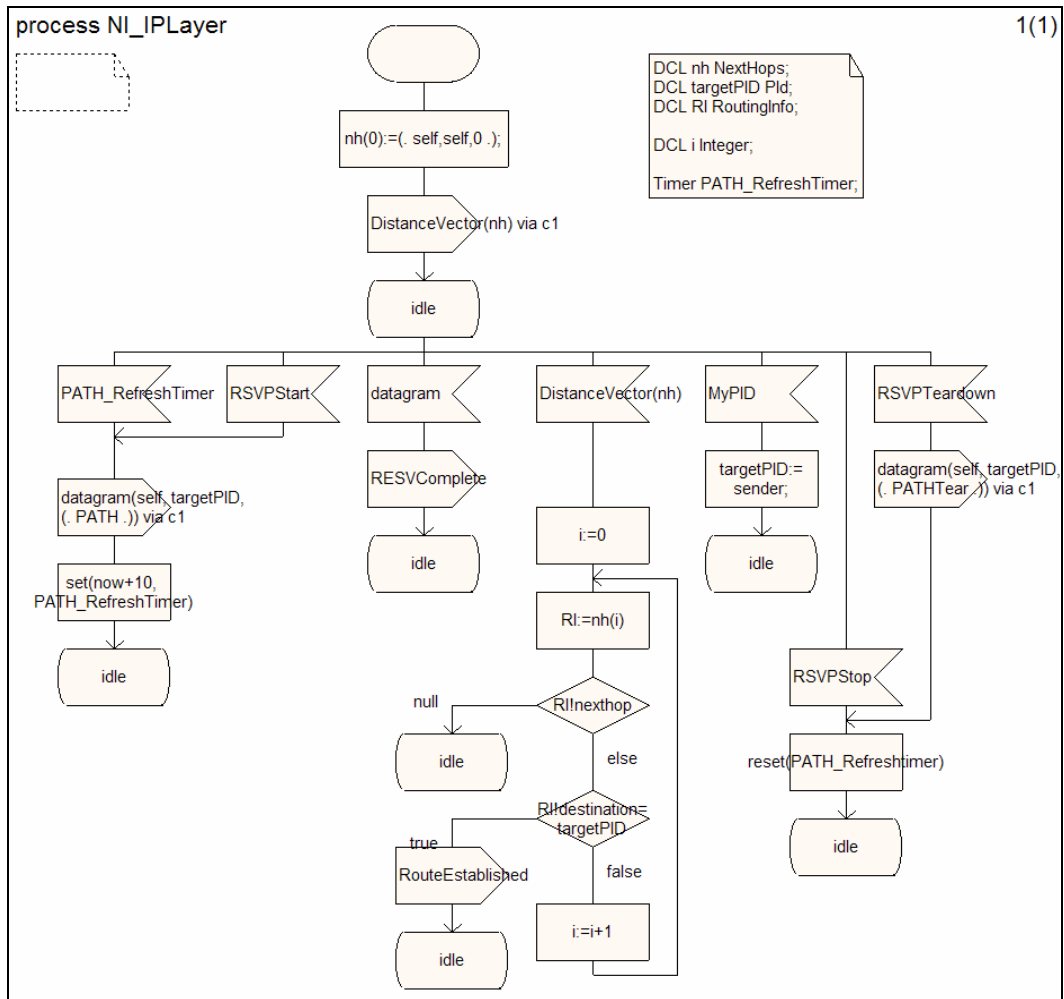


Notice that the remaining page 4 of the above Routing process description is empty and not shown here.

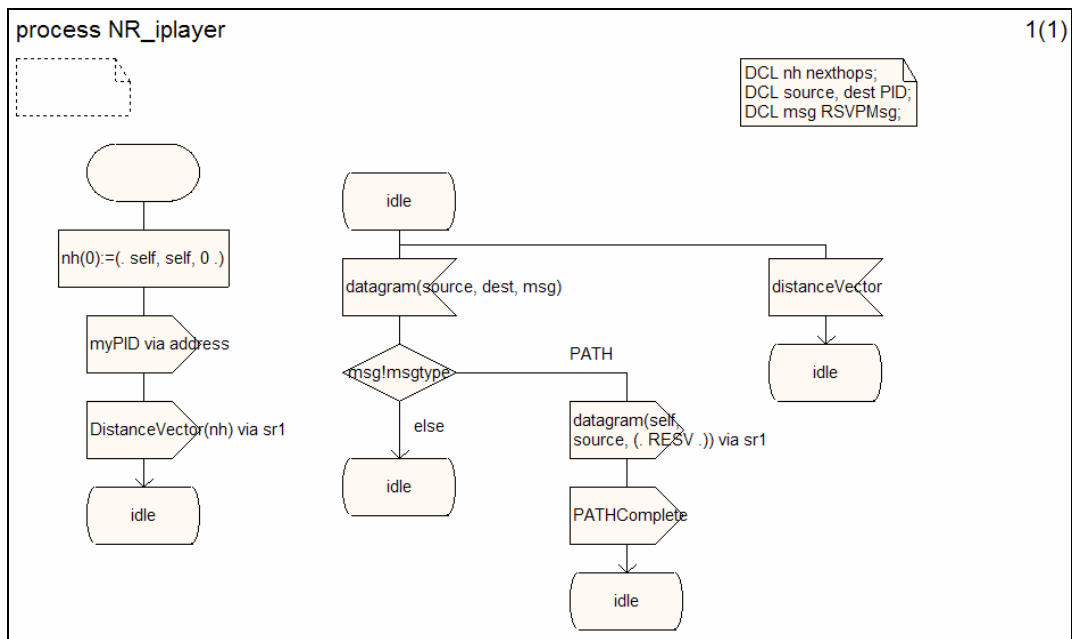
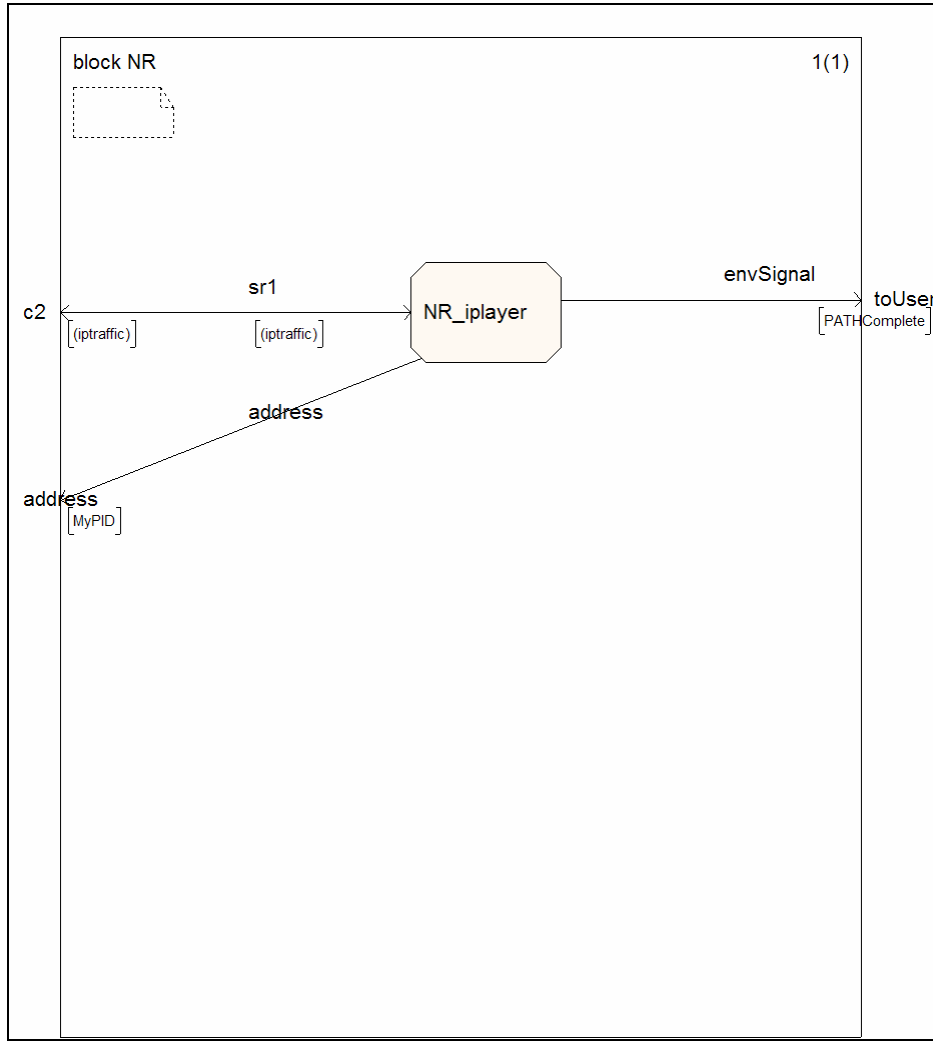


NI BLOCK AND PROCESS DIAGRAMS





NR BLOCK AND PROCESS DIAGRAMS



Curriculum Vitae

Constantin Werner

Geboren 26. November 1974 in Salzgitter-Bad

Staatsangehörigkeit deutsch

Wissenschaftlicher Werdegang

1981-1985 Grundschule Berg-Schule in Eppstein-Vockenhausen

1985-1988 Gesamtschule Freiherr-vom-Stein in Eppstein/Taunus

1988-1994 Christian-von-Dohm Gymnasium in Goslar; Abschluss: Allgemeine Hochschulreife

1998-2002 Studium der Informatik an der Technischen Universität Clausthal; Abschluss: Diplom-Informatiker

2000-2002 Studentische Hilfskraft am Institut für Informatik der Technischen Universität Clausthal

seit 01/2003 Wissenschaftlicher Mitarbeiter und Doktorand am Institut für Informatik der Georg-August-Universität zu Göttingen