Results in Using the New Version of the SDL-UML Profile

Alexander Kraas, Patrick Rehm

Fraunhofer Institute for Communication Systems (ESK)
Hansastr. 32
80686 Munich, Germany
{Alexander.Kraas, Patrick.Rehm}@esk.fraunhofer.de

Abstract: In the last two decades the Specification and Description Language (SDL) has been established in the telecommunication sector as a domain-specific modeling language. A disadvantage of SDL is the fact that only a small number of modeling tools for SDL is available. Another graphical modeling language is the Unified Modeling Language (UML) used for the model-based development of software systems. Due to the extensibility mechanisms of UML, this language is not only restricted to the software sector, but can also be aligned to other technical domains. In contrast to SDL, UML is supported by a large number of tools from different vendors. In order to make it possible to specify communication systems with UML, the International Telecommunication Union (ITU) has specified a UML profile for SDL. The first version of this profile was published in the year 1999 as recommendation Z.109. A completely revised version of this standard was prepublished in June 2007. This paper presents a first case study in applying the new standard version for modeling a communication system. Furthermore, advantages and open issues of the new standard version are also discussed.

Keywords: SDL-2000, UML2.1, XMI, OCL2.0, Back-to-Back User-Agent, SIP, Case Study

1 Introduction

On the one hand, for the model-based development of communication systems the Specification and Description Language (SDL) can be used. This standard is maintained by the International Telecommunication Union (ITU). New versions of the SDL standard are published periodically, whereas the latest version is SDL-2000 [2]. On the other hand, software systems can be modeled with the Unified Modeling Language (UML), in the latest version UML 2.1.2 [6], which is developed and maintained by the Object Management Group (OMG). In order to exchange UML models between tools from different vendors, the XMI [8] standard can be used for this purpose.

Owing to the fact that UML becomes more and more popular, not only in the software industry, several UML profiles have been specified for different technical domains, e.g. the AUTOSAR profile [12] for the automotive sector. The ITU has also recognized the potential of UML for the modeling of communication systems. Hence, a first version of a UML1.3 profile for the combined use of SDL and UML was published as recommendation Z.109 [1] in the year 1999. In this recommendation not only the specification of the profile is included, but also a mapping from UML stereotypes to corresponding SDL elements is defined. By applying those stereotypes to UML elements, only structural aspects and data types of a SDL system can be modeled. The specification of behavioral aspects within a UML model is not explicitly supported. The second edition [3] of the Z.109 recommendation was published by the ITU in 2007. In contrast to the first edition, the new Z.109 version specifies a UML2.0 profile which includes also stereotypes for the specification of behavioral aspects. This enables a seamless modeling of structural as well as behavioral aspects of a SDL system in UML.

The new possibilities of the second Z.109 edition have inspirited us to analyze the applicability of the SDL-UML profile for the UML-based specification of a complex SDL system. For this purpose we have chosen a Back-to-Back User-Agent (B2BUA) example which is a specialized element within a Session Initiation Protocol (SIP) [11] environment. A challenge was the lack of tool support for the new version of the SDL-UML profile. Therefore, we had to construct a toolchain which was suitable for our requirements. Due to the fact that the focus of our research activities is on the model-based test generation, we have not regarded the mapping of UML elements to their corresponding SDL-2000 elements. Instead of this, attention has been paid to the modeling capabilities of the SDL-UML profile. In future, models on which this profile has been applied shall serve as an additional input format for our model-based test case generation toolchain [10]. Apart from the modeling aspects, we have also specified appropriated OCL constraints for the SDL-UML profile, because in the Z.109 recommendation constraints are defined only textual.

The rest of this paper is structured as follows. Section 2 discusses other related UML profiles for modeling communication systems. In section 3 a case study and information about the required tools are presented. The use of OCL [6] constraints in combination with the SDL-UML profile is discussed in section 4. Our experiences which we have made until now by applying the profile can be found in section 5. The last section provides a brief summary and an outlook on our future work.

2 Related Work

Apart from the already mentioned two version of the Z.109 recommendation, also other approaches for the UML-based specification of communication systems were proposed. At the European Telecommunication Standardization Institute (ETSI) the working group STF-250 had been working on this topic until May 2004. As a result of the work a specification for a *„UML Profile for Communicating Systems“* [5] was presented. Unfortunately, this specification was incomplete and the work on the profile was discontinued. However, another profile [4] with the same name (UML-

CS) as the ETSI's one was originated by the Telematics Group of the University of Göttingen. In comparison to the SDL-UML profile of the ITU, the approach of the Telematics Group introduces some additional modeling concepts which can not be directly mapped to SDL. These concepts are especially dedicated for the modeling of internet protocols. A further difference between both profiles is the specification of OCL constraints for the UML-CS profile, whereas the SDL-UML profile only defines textual constraints.

As a matter of principle, instead of using the new version of the SDL-UML profile for our work, we could also use the UML-CS profile. But we have chosen the SDL-UML profile, because it rests on the standardized Z.109 recommendation, while the other one is not standardized.

3 Case Study and Required Tools

Before the model for the case study could be created, in a first step a convenient modeling tool chain had to be arranged. This was necessary due to a lack of tool support for the SDL-UML profile at the moment of writing this paper. A brief overview of the utilized tools and their necessary adaptations is given in the first part of this section. In the second part the Back-to-Back User-Agent (B2BUA) case study is discussed in detail.

3.1 Utilized Modeling Tools

Most of the following requirements are derived form the Z.109 recommendation, as postulated in section 1.1 – "Conformance". However, since we have analyzed UML modeling aspects only, SDL relevant requirements have not been considered for the tool selection.

Tool Requirements
Since SDL-UML is defined as a UML2.0 profile, an appropriated modeling tool has to implement the UML2.0 Superstructure. Furthermore, the utilized tool has also to support the usage of profiles and stereotypes. Without such a feature, it is infeasible to apply the SDL-UML profile on a UML model. Additionally, the Z.109 recommendation requires that a tool supporting the SDL-UML profile should be able to import/export models from/to other tools. The standard exchange format for UML models is the XML Metadata Interchange (XMI) [8] which is tightly coupled to the corresponding Meta Object Facility (MOF) [9]. This is the reason that UML2.0 models can only be exchanged with the XMI2.0 format, but not with older XMI versions. Therefore, a further requirement is the support of XMI2.0.

Apart from the requirements postulated explicitly in the standard, specified modeling constraints has also to be validated against the model. Due to the fact that the SDL-UML profile does provide constraints only as textual information, OCL [6] constraints were specified according to the given textual constraints. These constraints can be used in a UML tool to validate the correctness of created models. Since our

intention was to specify OCL constraints for each stereotype, contained in the SDL-UML profile, an appropriated tool have to support the specification and processing of constraints on UML metamodel level (M2). An example for the <<ActiveClass>> stereotype is provided in section 4.

Toolchain Architecture

A convenient modeling tool which can be used for the specification of SDL-UML compliant models has to fulfill the already mentioned requirements. Therefore, we have evaluated different UML2.0 modeling tools. Great disadvantages of the most considered tools were a missing support for XMI2.0 model interchange and only a restricted support for UML profiles. Furthermore, only a small number of those tools have supported OCL constraints on metamodel level. For our modeling tasks, we have selected MagicDraw [13]. As a matter of principle, this tool is also capable to validate OCL2.0 constraints which are specified for each stereotype of the SDL-UML profile. Due to some limitations related to the syntax check of OCL expression and some other problems concerning OCL constraints for properties of stereotypes, we have made the decision to use a further tool for the constraint validation. Hence, the Eclipse framework has been chosen for this purpose, because it provides plug-ins which support OCL and UML.

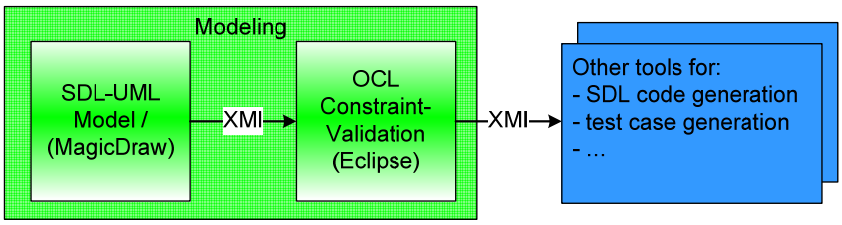

Fig. 1. Modeling toolchain architecture

Fig. 1 shows the architecture of the prototypical toolchain for creating and validating SDL-UML compliant models. As determined by the already mentioned requirements, the model interchange between the different components rests on the XMI format. Since the MagicDraw tool and the Eclipse framework only support the latest XMI2.1.1 standard, this version has to be used instead of XMI2.0. Furthermore, both tools implement the UML2.1.2 Superstructure instead of the required 2.0 version. The mentioned circumstances have had no further effects on the functionality of the toolchain.

Besides the built-in features of the Eclipse OCL plug-in, an additionally plug-in was implemented in order to validate automatically OCL constraints of the SDL-UML stereotypes. Apart from the “Modeling” components, it is conceivable that in future additional components, e.g. for SDL code or test case generation, are added to the prototypical toolchain.

3.2 Case Study: SIP System

In the following section a case study in applying the new version of the SDL-UML profile is provided on the example of a SIP Back-to-Back User Agent. For the sake of clarity the same conventions for the notation of the different kinds of modeling elements are used as defined in the Z.109 recommendation.

- A name written within double pointed brackets (<< … >>) refers to a stereotype of the SDL-UML profile.
- An underlined name refers to an element of UML Superstructure [7].
- SDL-2000 elements [3] are referred by names written in *italic* style.

The first approach to the Z.109 profile was the modeling of a SIP [10] System containing two User Agents, one Back-to-Back User Agent and a Registrar Server. Since this system has already been designed in SDL, not only the SDL-UML model could be compared with the SDL model, but also the methodology used to design those two models could be matched against each other. The first step in both approaches was to create the overall structure of the system.

Modeling of Structural Aspects and Data Types

In SDL models data types are specified in the same diagrams and context as the structure, while the new SDL-UML Profile separates those two modeling domains. **Fig. 2** shows the layout of the SIP User Agents and data-constructs of the SIP-System. Note that a Class stereotyped with an <<ActiveClass>> represents an *SDL Agent* and the Stereotype <<PassiveClass>> maps to the concept of a *SDL Abstract Data Type*. These two stereotypes are concrete subtypes of the abstract <<Class>> Stereotype, as defined in the Z.109 standard. However, some elements contained in the shown classes are omitted in the figure. The declaration of signals and parameters of the classes for example, are hidden for the sake of simplicity.

The tagged value “isConcurrent” indicates that the <<ActiveClass>> contains other active classes, which will be executed concurrently. If this value is set to “true”, a <<ActiveClass>> Class is mapped to a *Block* reference, otherwise to a *Procedure* reference. The main part of the designed SIP-System is the <<ActiveClass>> UserAgent. This class contains the SIP specific functions and the other (“Non-SIP”) functionalities of a SIP-Terminal. The “Non-SIP” function blocks implement mechanisms used for the communication with the system environment, whereas the SIP specific function blocks realize the layered design of a SIP User Agent. The <<PassiveClass>> classes are used to represent the SIP-Messages with their large headers, which are impossible to be modeled as single signal parameters. Instead, they are encapsulated in classes which can be specified as parameters.

The used concepts of object orientation are also shown in **Fig. 2**. For example, the Back-to-Back User Agent is only a redefined inheritance of a “normal” User Agent and the same applies to the registrar server. The classes “SIP_Header” and “SIP_Startline” are only abstract super-classes, which are altered to the concrete classes of the different message types. However, this class model does neither describe the structural relations and communication channels between the system components, nor are ports or gates of the SIP User Agent defined.

These important aspects of a communication system are shown in **Fig. 3**. It shows the internal structure of the SIP-specific block of the <<ActiveClass>> “UserAgent”

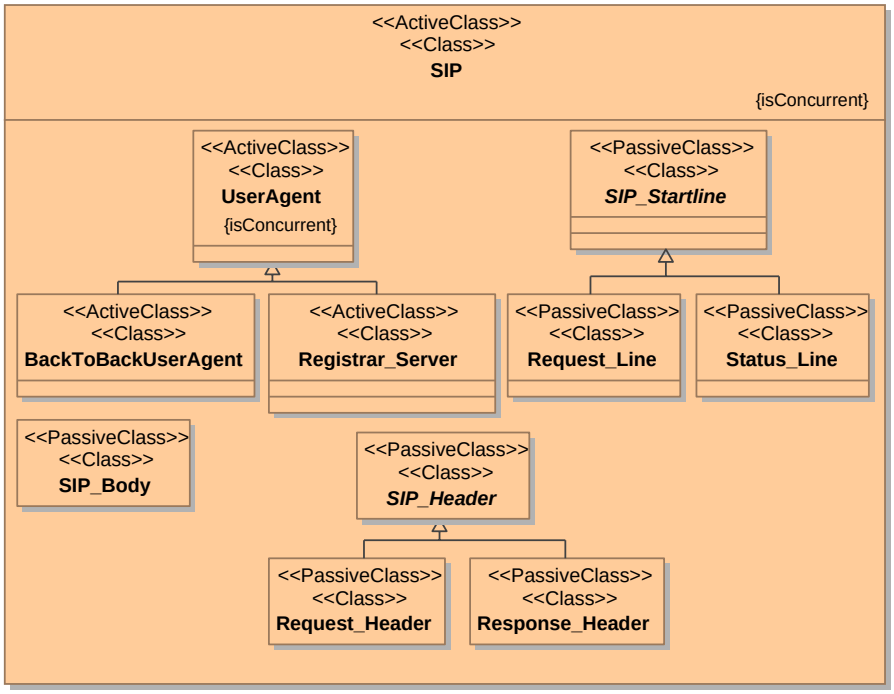

Fig. 2. The Class Overview of the SIP-System

The “initialNumber” tagged values are used to determine the initial number of object instances at the start of execution. According to the SIP standard [11], transaction instances are only created by the “Transaction User” during runtime.

The other parts of the diagram are similar to the common SDL notations. *SDL Channels* are defined in SDL-UML as communication paths between the instances and can be delayed or not. The signals that can traverse these channels are also defined here, but they are omitted in the example. In order to ensure a correct mapping to SDL, all UML elements are stereotyped in compliance to the SDL-UML profile.

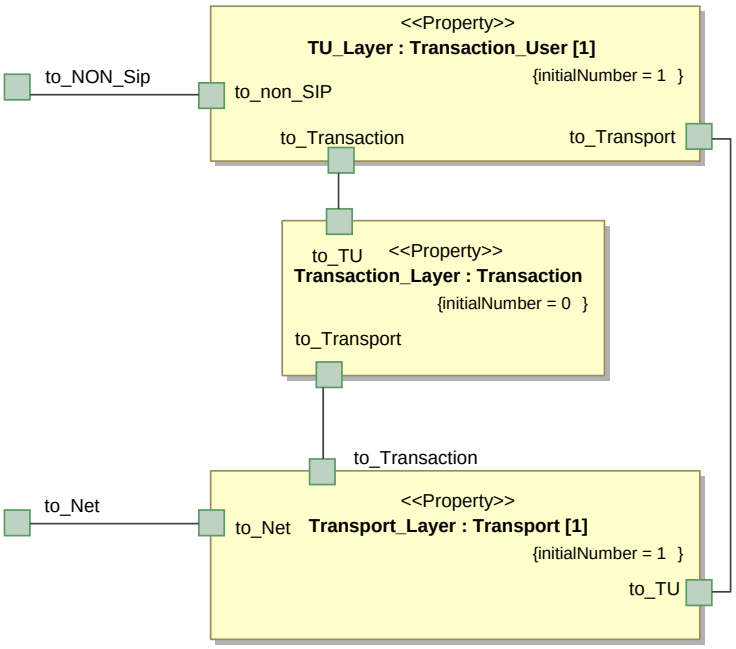

Fig. 3. The internal structure of the SIP User Agent

Modeling of Dynamical Behavior

The main difference between a native SDL model and a corresponding SDL-UML model is the methodology to specify the behavior of the system. In SDL behavioral diagrams which represent "Extended Finite State Machines" (EFSM) are used to characterize all dynamical aspects of a system in one modeling layer. The only way to organize such an EFSM in a hierarchical manner is to use *SDL Procedures* that encapsulate parts of the behavior. On the other hand, SDL-UML introduces a new level of abstraction in the modeling process. The StateMachines in the SDL-UML approach contain only state transitions and basic decisions, as shown in Figure 3. An Activity, which is executed during a state Transition, is specified as its effect.

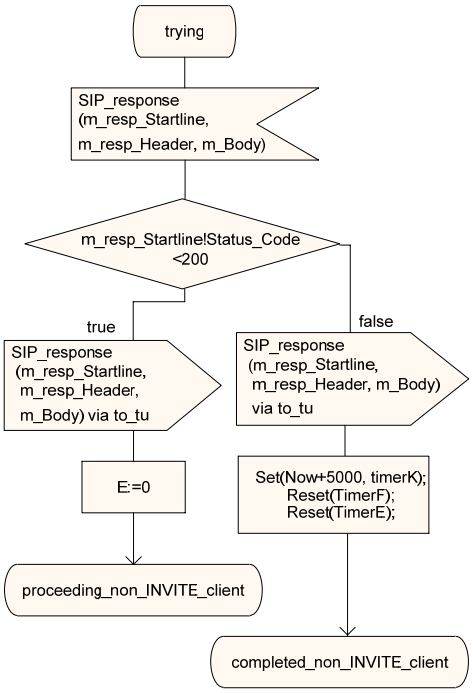

Fig. 4. SDL State Machine

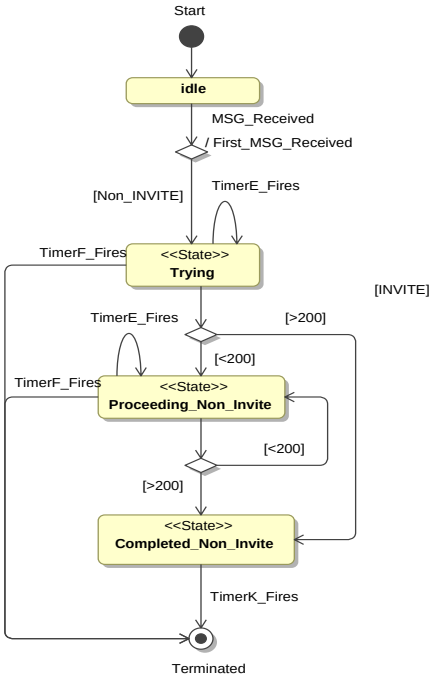

Fig. 5. SDL-UML State Machine

Fig. 5 shows an extract of the UML StateMachine (including applied SDL-UML stereotypes) which specifies the transaction layer of the SIP-User Agent in the case of a "Non-Invite-Client" transaction. **Fig. 4** shows a classical *SDL State Machine* of the same functional block. The most notable difference between both kinds of diagrams concerns transitions which are not any longer stimulated by direct input symbols, like in the SDL case. Instead, in SDL-UML models they are triggered by SignalReceiveEvents which are defined as the trigger of a Transition. Additionally, Transitions can be protected by guards, which must be fulfilled in order to enable the transition to be fired.

Decisions which are depending on, for instance, the current value of a variable are represented in SDL-UML StateMachines by Pseudostate choice elements, which correspond to the *SDL Decision* elements. When the system reaches such a Pseudostate, the guards of the outgoing Transitions determine the next step in execution.

The Z.109 recommendation specifies several rules, how guards and message triggers must be defined in order to allow a proper mapping to SDL elements. In order to be standard compliant, the StateMachine must not contain any other UML elements, than those that are required to perform state transitions. Neither VariableActions nor SendSignalActions are allowed here.

The comparison of **Fig. 4** and **Fig. 5** shows clearly that the SDL-UML StateMachines provide a much higher abstraction level of the system behavior, because actual activities are encapsulated inside the transition's effect.

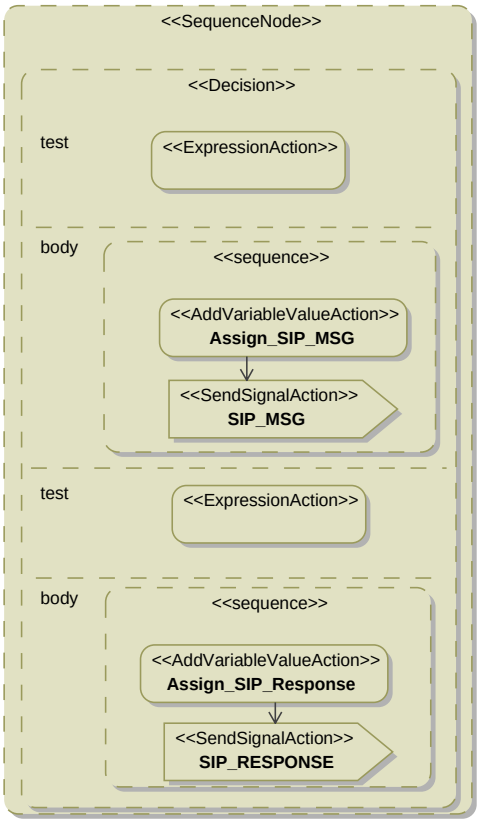

Fig. 6. Example for activity during a state change

The next step in modeling a SDL-UML system is the definition of UML actions for an Activity used as the Transition's effect, which occurs during a state transition. **Fig. 6** shows a basic example of how this can be realized. Owing to rules specified in the Z.109 recommendation, most of the UML elements in Activities have to be encapsulated in SequenceNodes. In the given example the SequenceNode contains a <<Decision>> node (ConditionalNode) with two outgoing paths. One path sends an SIP-Response and the other path sends a SIP-Request.

However, the designer has acquired an additional degree of freedom with this method, since he can decide if such basic decisions are to be made in the StateMachine layer by introducing a new transition or in the Activity layer. Like most of the additional parameters of all elements, the target or the path of outgoing signals have to be defined in the signal properties. Generally, all actions and tasks that a *SDL System* performs, while changing into a new state, are coupled to such Activities and SequenceNodes.

Future Topics
An open work item which will be finished next is the verification of the SDL-UML model using constraints defined within the profile, but in the following section a short outlook on the results of this work is given. Furthermore, a mapping of the SDL-UML model to SDL code which can be used to generate simulators or executable code to verify the model is also conceivable. Unfortunately, no tool which implements such a feature is available at the moment. But the presented case study will be checked against the OCL constraints in the near future, when the corresponding work item is finished.

4 OCL Constraints for the SDL-UML Profile

In section 1 it was already mentioned that constraints for UML stereotypes of the SDL-UML profile are specified in the Z.109 recommendation only as textual information. These constraints are inconvenient for an automatic validation against a SDL-UML model. In order to remedy this situation, convenient OCL2.0 constraints which can be automatically validated by a UML modeling tool have been specified. Since no standalone tool could fulfill the already mentioned requirements (s. section 3.1), a modeling toolchain was constructed and the Eclipse framework [14] was chosen for the OCL constraint validation.

Limitations of OCL2.0 constraints
The Object Constraint Language in its latest version 2.0 [6] is tightly coupled to the UML2.0. As a matter of principle, OCL constraints can be used on different modeling levels of the UML, for instance on UML metamodel level (M2). Since stereotypes reside on the M2 model level also, a tool which evaluates OCL constraints of stereotypes against a SDL-UML model have to support such a feature. In particular, attributes and associations of UML Metamodel elements, including properties of stereotypes, have to be accessible by OCL expressions.

A problem exists for accessing properties of stereotypes, because neither the UML [7] nor the OCL [6] standard define rules therefore. UML tools can only solve this problem in a proprietary way. This leads to the fact that OCL expressions including properties of stereotypes are tool-dependent. Usually, such constraints can not be interchanged without any adaptations between different UML tools.

<<ActiveClass>> Example
Owing to the fact that the Z.109 recommendation specifies approximately 150 different textual constraints for all stereotypes contained in the SDL-UML profile, it is nearly infeasible to depict them in this paper. Furthermore, until now we could not verify all OCL constraints, so only an example for the <<ActiveClass>> stereotype can be provided in **Table 1**. Besides other kinds of constraints, OCL also provides invariant constraints which are OCL expressions of type Boolean and they are associated to UML Classifiers. When an invariant constraint is evaluated, its result must be “true” otherwise the constraint is not fulfilled. All defined OCL constraints for the SDL-UML profile are of kind invariant.

Due to the already mentioned lack of standardization for accessing properties of stereotypes with OCL constraints, the invariant constraints in **Table 1** are defined on an abstract level. Following assumptions are made:

- Properties of a stereotype can be accessed in the same manner as attributes and associations of UML Metamodel elements.
- If it is necessary to refer to a stereotype, its name is written within double pointed brackets (<< … >>). In particular, this can be necessary for checking the type of an addressed UML element (`oclIsTypeOf`) or for type casting (`oclAsType`).

Table 1. OCL invariant constraints for the <<ActiveClass>> stereotype

1. An <<ActiveClass>> Class shall have isActive: Boolean = true
`self.isActive = true`
2. If isConcurrent: Boolean is false, isConcurrent of **any** contained instance shall be false.

`self.isConcurrent=false implies`
`  self.nestedClassifier`
`  ->forAll(oclAsType(<<ActiveClass>>).isConcurrent=false)`

3. If the <<ActiveClass>> Class has a classifierBehavior: Behavior [0..1], it shall be a StateMachine.
`self.classifierBehavior->notEmpty() implies` `  self.classifierBehavior->forAll(oclIsTypeOf(StateMachine))`
4. If an <<ActiveClass>> Class has a classifierBehavior: Behavior [0..1] and it has a superClass: Class [*] that is another <<ActiveClass>> Class that also has a classifierBehavior, the Statemachine of the sub-class shall redefine the Statemachine of the superClass. The reason is that in SDL the state machines of agents automatically extend each other, whereas this is not the case in UML.

`let sClass:Class = self.superClass->asSequence()->at(1)`
`in (self.classifierBehavior->notEmpty() and sClass->notEmpty())`
`  implies`
`  (sClass.oclIsTypeOf(<<ActiveClass>>) and`
`  sClass.classifierBehavior.oclIsTypeOf(StateMachine) and`
`  self.classifierBehavior.redefinedBehavior`
`  ->includes(sClass.classifierBehavior)`
`  )`

5. An <<ActiveClass>> Class used as the type: Type [0..1] of a composite property object (of another <<ActiveClass>> Class) shall have isAbstract = false (that is a typebased agent in an agent type shall not be based on an abstract type).

`self.part->notEmpty() implies`
`  self.part->select(oclIsTypeOf(Property)).type`
`  ->select(oclIsTypeOf(<<ActiveClass>>))`
`  ->forAll(oclAsType(<<ActiveClass>>).isAbstract=false)`

6. An ownedAttribute : Property [*] that has a type that is an <<ActiveClass>> Class and where aggregation == composite **shall not** have public visibility (an agent instance set cannot be made visible ouside the enclosing agent type).

```
self.ownedAttribute->notEmpty() implies
  self.ownedAttribute
    ->select(oclIsTypeOf(Property))
    ->select(type.oclIsKindOf(<<ActiveClass>>))
    ->select(aggregation=AggregationKind::composite)
      ->forAll(visibility<>VisibilityKind::public)
```

7. A nestedClassifier: Classifier [*] **shall not** have public visibility (an agent type, data type, interface type or signal definition cannot be made visible outside the enclosing agent type).

```
self.nestedClassifier->notEmpty() implies
  self.nestedClassifier
    ->forAll(visibility<>VisibilityKind::public)
```

8. An ownedConnector **shall not** have public visibility (a channel cannot be made visible outside the enclosing agent type that owns the channel).

```
self.ownedConnector->notEmpty() implies
  self.ownedConnector
    ->forAll (visibility<>VisibilityKind::public)
```

9. An ownedPort shall have public visibility (gates are visible ouside the enclosing agent type).

```
self.ownedPort->notEmpty() implies
  self.ownedPort->forAll(visibility=VisibilityKind::public)
```

10. An ownedBehavior: Behavior [0..*] shall **not have** public visibility (a procedure or composite state type cannot be made visible outside the enclosing agent type).

```
self.ownedBehavior->notEmpty() implies
  self.ownedBehavior
    ->forAll(visibility<>VisibilityKind::public)
```

11. An ownedBehavior: Behavior [0..*] shall only contain a StateMachine.

```
self.ownedBehavior->notEmpty() implies
  self.ownedBehavior->size()=1 and
  self.ownedBehavior->forAll(oclIsTypeOf(StateMachine))
```

Discussion

The above shown <<ActiveClass>> example proves that OCL can be principally used in order to express constraints for stereotypes of the SDL-UML profile. On the one hand, OCL constraints are more precise than textual specified constraints. On the other hand, complex textual constraints can also induce complex OCL expressions, e.g. constraints 4 and 5 in the example.

However, owing to the fact that OCL constraints are more precise than their corresponding textual constraints, it should be deliberated, if OCL could be a more

convenient alternative as the textual ones. Additionally, some UML modeling tools are able to check OCL invariant constraints not only on demand, but also during the creation time of a model which means that every time when the model is altered, associated constraints are validated automatically. For the model designer such a feature can be very helpful, because in the case of an error the designer can react directly.

5 Advantages and Open Issues

After providing a case study and showing the usage of OCL invariant constraints in the last two sections, advantages and open issues are briefly discussed here. But since our work is ongoing, not all topics can be covered completely.

Advantages of the new SDL-UML profile
One of the big advantages the new SDL-UML profile provides is clearly shown, when Fig. 4 and Fig. 5 in section 3.2 are compared to each other. The UML StateMachines used to model the system provide a much better overview of the system's overall behavior, than the "one in all" State Machines in SDL did. This is, of course, very beneficial to modern design paradigms like modularity. Different model designers can work on different parts of a model at the same time.

SDL-UML also benefits from the fact that various UML modeling tools from different vendors are available. Between these tools models can be interchanged in the XMI format. In the case of SDL there exist only two major commercial modeling tools from different vendors. Furthermore, some concepts of SDL have not been implemented in both tools until now, whereas some vendor-specific features can be used.

Another major advantage of the SDL-UML profile, however, is the improved split-up between the different parts of a model. In SDL everything was basically designed in one big diagram for every layer, including structure, data and behavior. The Z.109 on the other hand utilizes all the benefits the UML design process offers. There are different diagrams for every part of the system. The data is modeled independent from the structure or the behavior and so on. Apart from UML elements required in SDL-UML models, also additional UML diagrams or elements can be contained in a system model. The additional information can be used for other purposes than SDL code generation, for example advanced test case generation or system analysis. This is a big advantage in contrast to existing SDL tools, because it is nearly infeasible to augment such legacy tools with new features.

Open Issues
As any new profile or new modeling technique, the SDL-UML profile also has some open issues. One major drawback will probably be solved over time, but owing to the fact that the profile is relatively new, there are neither adequate modeling tools available for modeling with the profile, nor is the profile itself available in an applicable form for modeling tools. Every developer that have to design a system using the SDL-UML profile has to implement the profile itself, which is quite an

arduous task when considering all the constraints. The fact that there are no specialized tools available only complicates any design process at the moment.

For example, in the provided case study, which has already been simplified a little bit, the UML StateMachine of the Transaction Layer in the SIP User Agent alone contains about 25 transitions, that all have to be filled with Activities, Triggers and Guards one by one. Also the design process itself is quite laborious, because every single detail has to be specified and modeled by hand. In order to improve the usability of SDL-UML, a proper tool could provide customizable features enabling the user to configure or to create diagram templates or user specific diagrams. With such a feature, for instance, Activity diagrams could be automatically created when the Effect of a Transition have to be specified.

Another open issue is the specification of constraints for the stereotypes contained in the SDL-UML profile. As stated in section 4, OCL invariant constraints are more formal than textual constraints. Additionally, the model designer can also benefit from UML modeling tools supporting the automatic validation of OCL constraints during the creation of SDL-UML models. Due to these reasons, we propose to consider whether OCL constraints could be a useful extension for the Z.109 recommendation.

6 Conclusion and Future Work

The Specification and Description Language (SDL) has been established in the telecommunication industry for a long time. Also the language itself has been improved and extended continuously. One of these extensions is the development of the SDL-UML profile in its second version, enabling the specification of SDL systems by means of UML modeling tools. In contrast to dedicated SDL modeling tools, this solution has the advantage that not only SDL-UML specific model elements can be used, but also additional elements. This can encourage the development of advanced tools and modeling methodologies which going beyond the scope of the native SDL.

The case study provided in this paper has shown that the SDL-UML profile is convenient for modeling complex communication systems, like the Back-to-Back User-Agent example. Owing to the fact that the profile has been introduced one year ago, the tool support until now is quite inadequate. As discussed in section 4, a feasible improvement could be the introduction of OCL constraints which are automatically validated during the model creation.

Since work on OCL constraints for the SDL-UML is still in progress, in future we will provide a complete set of constraints for the profile. Also we consider to improve our developed plug-in for the Eclipse framework which makes it possible to validate the OCL constraints automatically against a SDL-UML model. Last but not least, we also plan to use SDL-UML models as input for our toolchain for automatic test case generation.

References

[1] International Telecommunication Union (ITU): SDL combined with UML, ITU-T Recommendation Z.109, November 1999
[2] International Telecommunication Union (ITU): Specification and Description Language (SDL), ITU-T Recommendation Z.100, August 2002
[3] International Telecommunication Union (ITU): SDL-2000 combined with UML, ITU-T Recommendation Z.109, June 2007
[4] Kraatz, S., Hogrefe, D., Werner, C.: A UML Profile for Communicating Systems. In: 5th International Workshop on System Analysis and Modeling, SAM 2006, LNCS, vol. 4320, pp. 1--18, Springer Verlag , 2006
[5] Telecommunication Standardization Institute (ETSI), STF-250: The UML Profile for Communicating Systems, 18. October 2004, http://portal.etsi.org/docbox/MTS/MTS/05-Meetings/ARCHIVE/2004/200410-MTS39/39TD28%20UMLCS%20book.pdf
[6] Object Management Group (OMG): OCL 2.0 Specification, Version 2.0, ptc/2005-06-06, June 2005
[7] Object Management Group (OMG): Unified Modeling Language (UML), Superstructure V2.1.2, formal/2007-11-02, November 2007
[8] Object Management Group (OMG): MOF 2.0/XMI Mapping, Version 2.1.1, formal/2007-12-01, December 2007
[9] Object Management Group (OMG): Meta Object Facility (MOF) Core Specification, Version 2.0, formal/06-01-01, January 2006
[10] Kraas, A. et al.: A generic toolchain for model-based test generation and selection. In Testing of Software and Communicating Systems: Work-in-Progress and Position Papers, Tool Demonstrations and Tutorial Abstracts of TestCom/FATES 2007, Tartu University Press, 2007, pp. 14-17
[11] Internet Engineering Task Force (IETF), RFC 3261 SIP: Session Initiation Protocol. June 2002
[12] AUTOSAR: UML Profile for AUTOSAR, Version 1.0.1, 27.06.2006 http://www.autosar.org/download/AUTOSAR_UML_Profile.pdf
[13] No Magic Inc., MagicDraw 15.1, Commercial UML modeling tool, http://www.magicdraw.com/
[14] The Eclipse Foundation, Eclipse 3.3.2, Open source software, http://www.eclipse.org/platform