Model-based Testing: Transforming SDL-UML models to the Intermediated Format 2.0

T Rogenhofer

26 October, 2009

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Model-based Testing: Transforming SDL-UML models to the Intermediated Format 2.0

A dissertation submitted in partial fulfilment of the requirements for the Open University’s Master of Science Degree in Computing for Commerce and Industry

Thomas Rogenhofer
(X5134702)

12 February 2010

Word Count: 14509
Preface

I wish to thank my supervisor Jonathan Stephenson for his support, advice and guidance throughout the entire research project. His responses were always encouraging and helpful.

I would also like to thank the many people at my employer the European Patent Office who have supported me all the way through my MSc studies. In particular I would like to express my gratitude to John Bambridge, Kevin Douglas, François Knauer and Thomas Placzek.

Finally, I would like to thank Alexander Kraas from the ESK Fraunhofer Institute for his support.
# Table of Contents

Preface ........................................................................................................................................................................................................... ii

List of Figures ................................................................................................................................................................................................... vii

List of Tables ............................................................................................................................................................................................ ix

Abstract ..................................................................................................................................................................................................... x

Chapter 1 Introduction ............................................................................................................................................................................. 1

1.1 Model-based Testing ........................................................................................................................................................................ 1

1.2 Abstract test case generation from system models ..................................................................................................................... 2

1.3 Tools supporting abstract test case generation .......................................................................................................................... 6

1.4 Modelling languages relevant for this research project .......................................................................................................... 7

1.5 The AGEDIS methodology and toolset .................................................................................................................................... 12

1.6 Summary of research problem ...................................................................................................................................................... 14

1.7 Aims and objectives of the research project ............................................................................................................................. 15

1.8 Overview of the dissertation ......................................................................................................................................................... 15

Chapter 2 Literature Review ............................................................................................................................................................... 17

2.1 Concepts of Model-based Testing ................................................................................................................................................. 17

2.2 The Intermediated Format 2.0 ......................................................................................................................................................... 19

2.3 UML 2.0 and the SDL-UML profile .......................................................................................................................................... 23

2.4 Syntax and semantics of the SDL-UML profile and IF 2.0 ....................................................................................................... 27

2.5 Model transformations ................................................................................................................................................................. 29
Appendix A: Extended Abstract................................................................................85
Appendix B - Result of analysis of SDL-UML modelling concepts.........................90
Appendix C - Results of comparative analysis ......................................................104
Appendix D - SDL-UML ticket machine model....................................................149
Appendix E - IF 2.0 ticket machine example with explicitly defined
             signalroutes..........................................................................................153
Appendix F – AGEDIS IF 2.0 ticket machine specification...................................155
Appendix G - MOFScript generated IF 2.0 code ..................................................201
Appendix H - MOFScript transformation template...............................................205
Appendix I - Abstract test case for verification method .......................................212
List of Figures

Figure 1 UML class diagram representing stack data structure.................................3

Figure 2 UML state machine diagram representing behaviour of instances of the
UML stack class........................................................................................................4

Figure 3 Abstract test case for incrementing the stack counter .................................5

Figure 4 Executable test case for incrementing the stack counter ..............................6

Figure 5 IF 2.0 system specification example ............................................................10

Figure 6 The AGEDIS architecture (modified from source: Hartmann and Nagin
(2004))..................................................................................................................13

Figure 7 IF 2.0 toolset architecture (modified from source: Bozga et al. 2004a).........20

Figure 8 Structure and communication between IF 2.0 processes (source:
modified from Graf et al. 2006b).........................................................................22

Figure 9 Relationship between UML 2.0 modelling capabilities and concepts ........25

Figure 10 Extension of UML metaclass by a stereotype.............................................27

Figure 11 Syntax and semantics in the SDL-UML to IF 2.0 transformation process.....29

Figure 12 Classification of model transformation approaches .................................30

Figure 13 MOFScript tool and the Eclipse Modelling Framework ............................31

Figure 14 Architecture of MOFScript tool (modified from source: Oldevik 2006)....32
Figure 15 The AGEDIS-TGV engine .................................................................35

Figure 16 Inputs to the MOFScript tool ..........................................................44

Figure 17 XSLT based transformation method ................................................45

Figure 18 Test directive for generation of an abstract test case....................49

Figure 19 UML Composite Structure Diagram for SDL-UML ticket machine
model .............................................................................................................54

Figure 20 SDL-UML ticket machine class diagram showing interfaces and signals ....55

Figure 21 <<StateMachine>> of <<activeClass>> TicketTray .........................56

Figure 22 <<Activity>> returnFalse for ReceiveSignalEvent EmptyTicketTray ....57

Figure 23 Definition of an IF 2.0 Signalroute .................................................62

Figure 24 MOFScript tool syntax checking functionality ..............................68

Figure 25 XMI elements of SDL-UML ticket machine model .......................69

Figure 26 IF 2.0 procedure call ....................................................................70

Figure 27 AGEDIS debugging information for IF 2.0 system specification from
Appendix E ...................................................................................................71
List of Tables

Table 1 Overview on SDL-UML modelling capabilities and underlying concepts .......39

Table 2 Overview on structure of comparative analysis results table.........................42

Table 3 List of UML 2.0 modelling capabilities used for the specification of the SDL-UML profile ................................................................................................................52

Table 4 Grouping of SDL-UML stereotypes and IF 2.0 modelling concepts ..............58

Table 5 Overview on verification results of transformation hypotheses....................73
Abstract

In 2007, the International Telecommunication Union published a SDL-UML profile. This SDL-UML profile enables the modelling of software systems with the Unified Modelling Language (UML) 2.0 according to semantics of the Specification and Description Language (SDL). Although it is possible to automatically generate abstract test cases from SDL models, this is not the case for SDL-UML models. This dissertation describes the work done to transform a SDL-UML model to an intermediate text based language, the so-called Intermediated Format (IF) 2.0. This transformation allows an existing automatic testing suite to generate test cases from an IF 2.0 system specification.

At the beginning of this research project a comparative analysis was performed in order to investigate to what extent the SDL-UML profile could be transformed to the Intermediate Format 2.0. The results of this analysis were used for the specification of a SDL-UML model that was transformed to an IF 2.0 system specification. This transformation was supported by an Eclipse-based transformation tool. An existing automatic test suite was then used for the verification of the SDL-UML to IF 2.0 transformation.

The verification of the transformation showed that all elements of the SDL-UML model except the one for the modelling of communication channels could be transformed to IF 2.0. For an automatically generated test case, these communication channels are required for the sending and receiving of signals from a system under test. As a result of this, the SDL-UML model could not be transformed to Intermediated Format 2.0 for the purpose of automatic test case generation.
Chapter 1 Introduction

In 2007, the International Telecommunication Union has published a SDL-UML profile (ITU-T 2007). This SDL-UML profile enables modelling of software systems with UML 2.0 according to SDL-2000 semantics. This dissertation describes the work done to transform a SDL-UML model to an intermediate text based language, the so-called Intermediated Format 2.0. This transformation allows an existing automatic testing suite to generate test cases from an IF 2.0 system specification.

This dissertation is embedded within the software testing discipline of Model-based Testing (MBT). The following chapters describe the background to the research problem and how aspects of the SDL-UML transformation relate to Model-based testing concepts and tool chains.

1.1 Model-based Testing

With the increasing complexity of software systems, the quality control has become a challenging task. One of the responses of the industry has been the development of Model-based Testing. MBT can have the benefits of reduced testing time and cost, increased fault detection, or traceability among requirements, system model and tests. In the last couple of years, considerable research and prototyping has now evolved into some commercial MBT tools. An example for this is the AGEDIS research project. It was funded by the European Union in order to increase the competitiveness and efficiency of the European software industry by integrating MBT into the software development process. The AGEDIS project delivered an applicable methodology and toolset for Model-
based Testing of distributed and reactive systems. Parts of this methodology were also integrated into commercial MBT tools. Examples of commercial MBT tools are the IBM Rational SDL Suite (IBM 2009), or the Eclipse based Conformiq Qtronic tool (Conformiq 2009). All these commercial MBT tools or scientific prototypes follow the same MBT methodology that can be described as follows.

Utting and Legeard (2007) define Model-based Testing as the generation of test cases with expected output values from a behavioural model of a system. They state that the MBT process can be abstracted and separated into the following five tasks:

1. A model of the system is created.
2. Abstract test cases are generated from this model.
3. Abstract test cases are concretized in order to make them executable.
4. Executable test cases are run against the system under test (SUT) with verdicts being assigned.
5. The test results are analyzed.

1.2 Abstract test case generation from system models

System models that are used for Model-based testing consist of a structural and behavioural part. The structural part of a model specifies system objects and their attributes. The behavioural part of a model specifies the behaviour of system objects. Software testers are often interested in the observable behaviour of a system, which contains inputs to and outputs from the system under test.

Pretschner and Philipps (2005) define sequences of observable behaviour as traces. Within the context of MBT, these traces are interpreted as test cases. A test case consists of inputs, so-called fixtures, and expected outputs, also referred to as
oracles (The Open University 2005). Executing a test case means that inputs are being fed into the SUT and outputs of the SUT are being compared to the oracles. Pretschner and Philipps state that a behavioural model can contain an infinite number of infinite traces. For this reason it is necessary to select a number of finite traces that can be translated into test cases. This selection is achieved by means of so-called test directives. They provide a number of selection criteria like state coverage of system objects, or transitions between states of system objects.

The following example illustrates the generation of an abstract and executable test case from a trace of a behavioural model. The UML class model (Figure 1) represents a stack data structure and its behaviour is modelled by means of a UML state machine (Figure 2). The test case for this example covers the trace of incrementing the counter value from two to three. The fixture of this test case is the attribute counter being equal to two. The expected output from the SUT would be that the attribute counter is equal to three. The trace, i.e. part of the observable system behaviour, for this test case includes the input to the SUT via operation call `Stack.increase()` and the output of the SUT via operation call `Stack.getCounter()`.

![UML class diagram representing stack data structure](image)

Figure 1 UML class diagram representing stack data structure
Figure 2 UML state machine diagram representing behaviour of instances of the UML stack class

The previously defined trace of incrementing the stack can be transformed into an abstract and executable test case. The advantage of an abstract test case notation is that it provides the same level of abstraction as the system model. This makes it easier to validate whether the abstract test case covers the behavioural trace as intended (Pretschner and Philipps 2005). Another advantage of an abstract test case notation is that it is platform independent. This enables the abstract test case to be made executable for more than one platform (e.g. Java, or C++).

An example of an abstract test case notation related to the stack system model is given in Figure 3. This abstract test case was generated manually and covers the trace of incrementing the counter value from two to three. This representation follows the same XML based abstract test case notation as used within the AGEDIS project (Nagin and Kirshin 2003). It is assumed for this abstract test case that the attribute `Stack.Counter` has the value 2 within an instance of the Stack class represented in Figure1.
Figure 3 Abstract test case for incrementing the stack counter

Utting and Legeard state that the transformation of an abstract test case into an executable test case represents the third task in the MBT process. Figure 4 provides an example on how the abstract test case from Figure 3 can be transformed into an executable test case. The executable test case in Figure 4 had been translated manually from the abstract test case in Figure 3. This executable test case illustrates how the test fixture, SUT output and expected output are realised in an applicable solution. It is important to note that the executable test case was written according to a JUnit notation. JUnit is a testing framework for Java programming code that provides a number of test classes and methods.

```java
public class TestStack extends TestCase {
    /*fixture
    private Stack stackA;
    private Stack stackB;
    /*fixture setup
    protected void setup() {
        stackA = new Stack(2);
        stackB = new Stack(2);
```
Figure 4 Executable test case for incrementing the stack counter

1.3 Tools supporting abstract test case generation

A direct generation of abstract test cases from a system model is not always possible. In such a case, it is necessary to transform the source model of the SUT into another target model. A common reason for model transformations in MBT is that abstract test case generators require a special input format.

An example for such an abstract test case generator is the TGV engine developed by VERIMAG and Irisa (Graf et al. 2006b). The TGV engine generates abstract test cases for conformance testing of information systems and protocols. An example for the application of the TGV engine was the generation of abstract test cases for the DREX protocol that represents a military version of the ISDN D protocol (Jard and Jerón 2005). Another example for the TGV engine based generation of abstract test cases was the European Transit Computerisation Project. This project had the objective to develop a system that would allow for the monitoring of goods in transit between EU countries.
According to Jard and Jerón (2005), the TGV engine requires two kinds of input. The first input is a formal specification of the SUT. This system specification can be formalised using one of the following modelling languages: the Unified Modelling Language, the Specification and Description Language (SDL), or the Intermediated Format (IF) 2.0. The second input to the TGV engine is a formal specification of test directives that is needed for the test case generation. The TGV engine requires test directives to be formalised in the Intermediated Format 2.0.

1.4 Modelling languages relevant for this research project

The modelling of a SUT for Model-based testing can be achieved through means of a modelling language that satisfies certain requirements. Utting and Legeard (2007) state that a modelling language must be precise enough in order to describe the behavioural aspects of a SUT. The level of precision of a modelling language is closely related to the underlying formalisms of this language. Examples of modelling languages that are precise enough for modelling behavioural aspects are UML, SDL, or IF 2.0. Each of these three languages can be applied for the following specific modelling domains.

The Intermediated Format 2.0 represents a modelling language that allows systems to be specified as “(...) a set of timed automata communicating asynchronously through a set of buffers or by rendezvous through a gate of synchronised gates” (Bozga et al. 1999). The Intermediated Format 2.0 was developed at VERIMAG (Graf et al. 2006b) together with a toolset that forms an architecture for IF 2.0 model validation and exploration (Bozga et al. 2000, 2004b). The objective for this development was to elaborate a toolset and intermediate language representation that would allow for simulation,
verification, testing and debugging of real–time system models. The previously introduced TGV engine is part of this IF 2.0 toolset architecture.

Although IF 2.0 has a simple syntactic structure and corresponding operational semantics (Bozga et al. 1999a), its concepts are sufficient for the modelling of timed asynchronous systems. One advantage of IF 2.0 is that its formally defined operational semantics can be utilised for model simulation tools or abstract test case generators like the TGV engine. Another advantage of IF 2.0 is that it is general enough to cover subsets of concepts from other languages like UML or SDL.

Figure 5 shows an example of an IF 2.0 ticket machine system specification with the following functionality. A user of the ticket machine can send the signal EmtpyTicketTray1 to the process TicketMachine via the signalroute sig0. The process TicketMachine then calls the process TicketTray in order to empty the ticket tray. In this example, the process TicketTray is in the state of ready, meaning that the ticket tray is not full and can therefore also not be emptied. For this reason, the process TicketTray returns the signal EmtpyTicketTray2(false) to the process TicketMachine. Upon receipt of this signal, the process TicketMachine informs the user that the ticket tray can not be emptied. This example of an IF 2.0 system specification illustrates the simple syntactic structure of the IF 2.0 language specification as described by Bozga et al. (1999a).

```plaintext
system TicketMachine;
signal EmtpyTicketTray1(boolean);
signal EmtpyTicketTray2(boolean);
```
signalroute sig0(1) from env to TicketMachine with EmtpyTicketTray1;
signalroute sig1(1) from TicketMachine to env with EmtpyTicketTray1;
signalroute sig2(1) from TicketMachine to TicketTray with EmtpyTicketTray2;
signalroute sig3(1) from TicketTray to TicketMachine with EmtpyTicketTray2;

process TicketMachine(1);

var success boolean;
var resultEmptyTicketTray boolean;

state start1 #start;
nextstate initialized;
endstate;

state initialized;
input EmtpyTicketTray1(resultEmptyTicketTray);
output EmtpyTicketTray2(resultEmptyTicketTray) via {sig2}0;
nextstate call_TicketTray;
endstate;

state call_TicketTray;
input EmtpyTicketTray2(resultEmptyTicketTray);
task success := resultEmptyTicketTray;
output EmtpyTicketTray1(success) via {sig2}0;
nextstate initialized;
endstate;
endprocess;

process TicketTray(1);

state ready #start;
input EmptyTicketTray1();
output EmptyTicketTray2(false) via {sig3}0;
nextstate ready;
endstate;
endprocess;
endsystem;

Figure 5 IF 2.0 system specification example

The Specification and Description Language was mainly developed for the modelling of distributed and reactive systems in telecommunications. SDL is standardised by the International Telecommunication Union (ITU) and has evolved over more than two decades (Mitschele-Thiel 2001). Major revisions of SDL have taken place in 1992 and 1999. These revisions have led to SDL-92 as specified in ITU-T (1993) and SDL-2000 as specified in ITU-T (2002).

The evolution from SDL-92 (ITU-T 1993) to SDL-2000 resulted in a number of changes that are thoroughly documented in the work of Holz (2001) and Mitschele-Thiel (2001). These changes can be summarised as the removal, alignment and introduction of new modelling concepts. In addition, SDL-2000 is now based on object-orientation and has new formal semantics (Holz 2001). In comparison to IF 2.0, SDL is specified in two syntactical forms: a textual and a graphical representation. One of the advantages of SDL is its formally defined operational semantics. This may lead to the assumption that SDL could be used for the intended transformation without the need for the Intermediated Format 2.0. The following discussion will show that this not possible for reasons related to the structure of test case generation rules.
The Unified Modelling Language can be seen as the de facto standard for modelling of object-oriented systems (Selic 2006). UML is standardised by the Object Management Group and has also evolved over nearly two decades. UML 2.0, as specified in OMG (2007a, b) was released as a major revision of UML 1.4 (OMG 2001). The major evolution from UML 1.4 to UML 2.0 can be summarised as follows:

1. Improved capability for the modelling of large-scale software systems.
2. Improved language-extension mechanisms for domain specific modelling.
3. Revision of UML 1.4 modelling concepts leading to the removal, modification or clarification of already existing modelling concepts.

The ITU used the UML 2.0 domain extension mechanism for the specification of a SDL-UML profile (ITU-T 2007). This SDL-UML profile enables the modelling of software systems with underlying SDL-2000 semantics. The advantage of the SDL-UML profile is that it becomes possible to model systems according to SDL-2000 semantics with the support of UML tools. The disadvantage of the SDL-UML profile is that no applicable solution exists for the automatic test case generation from SDL-UML models.

This is not the case for systems being modelled in SDL-92. The automatic test case generation from SDL-92 models has been possible for many years. With respect to this, Belinfante et al. (2005) provide examples of commercial tools that enable automatic test case generation from SDL-92 models. It could be assumed that the automatic test case generation from SDL-UML models can be achieved by reusing the existing SDL-92 test case generators. This is not possible for the following reasons.
The evolution from SDL-92 to SDL-2000 contains major changes like the introduction of new modelling concepts, or the modification of already existing modelling concepts. These new SDL-2000 concepts can not be translated into test cases by the existing SDL-92 test case generators. This problem can be solved through adaptation of the AGEDIS project methodology and toolset. Before describing how this can be achieved, the AGEDIS project shall be described in more detail.

1.5 The AGEDIS methodology and toolset

As mentioned at the beginning of this chapter, the AGEDIS project had the objective to deliver a MBT methodology and supporting toolset.

Figure 6 provides an overview on the AGEDIS software components, their inputs and the public interfaces.
According to Hartmann and Nagin (2004), the AGEDIS methodology can be described as follows. In a first step, the AGEDIS modelling language (AML) is used for specifying a system model. The AML represents an UML profile that is based on UML 1.4 (Cavarra et al. n.d.). In a next step, the AML model is transformed into the Intermediated Format 2.0. Once this transformation has been completed, the IF 2.0 specification together with test directives is input to the TGV engine (Jard and Jerón 2005). Then, the TGV engine generates abstract test cases according to test directives. In a final step, the abstract test cases are translated into executable test cases and run against the SUT.

The AGEDIS methodology is supported by a number of software components that have been integrated into the AGEDIS toolset. Two important software components of this toolset are the AGEDIS compiler and the TGV engine.
AGEDIS compiler transforms the AML system model into the IF 2.0, which is then input into the TGV engine.

The AGEDIS toolset and methodology can be utilised for the generation of SDL-UML test cases, whereby the following needs to be considered. According to the AGEDIS methodology, models that are input to the AGEDIS compiler must be specified in the AGEDIS modelling language (Cavarra and Davies 2001). The AGEDIS modelling language represents an UML profile that is based on UML 1.4 (Cavarra et al. n.d.). As a consequence of this, the AGEDIS compiler can not transform SDL-UML concepts to IF 2.0 for the following reason.

SDL-UML is based on UML 2.0 and contains modelling concepts that can not be interpreted by the AGEDIS compiler. Solving this issue with a transformation of SDL-UML to AML is also not possible for the following reason. The semantics of the SDL-UML profile is based to a large extent on UML 2.0 modelling concepts that enable the modelling of composite system structures. Examples for these UML 2.0 modelling concepts are the concepts of Port, Connector, or StructuredClassifier. These UML modelling concepts do not exist in the AML language definition. Therefore, the mapping of these SDL-UML modelling concepts to the AML profile is not possible without a considerable loss of SDL-UML semantics.

1.6 Summary of research problem

The discussion on the AGEDIS toolset showed that the AGEDIS compiler can not be used for SDL-UML to IF 2.0 transformations. Nevertheless, the AGEDIS architecture is flexible enough to provide a solution for this problem. Within the
AGEDIS architecture, IF 2.0 specifications can be input directly to the TGV engine. If it were possible to transform SDL-UML concepts to IF 2.0, then the automatic generation of SDL-UML test cases could be possible with support of the AGEDIS toolset and methodology.

### 1.7 Aims and objectives of the research project

The aim of the research project is to investigate whether it is possible to transform SDL-UML models to models of the Intermediated Format 2.0 for the purpose of automatic test case generation.

Given this aim of the research project, the following objectives have been set for performing the research:

1. To analyse to what extent modelling capabilities and underlying concepts of the SDL-UML profile share similarities with those of the IF 2.0.
2. To investigate reports of practice in research or industry of UML 2.0 (respectively UML 1.4) and SDL-2000 (respectively SDL-92) to IF 2.0 transformations.
3. To investigate a methodology that can be reused for the transformation of SDL-UML to IF2.0.
4. To define an architecture of software components that supports the transformation methodology and the generation of abstract test cases.
5. To develop transformation rules for the transformation of SDL-UML to IF 2.0.
6. To prove the hypothesis of SDL-UML to IF 2.0 transformations by means of manual and automatic test case generation.

### 1.8 Overview of the dissertation

This dissertation continues with a literature review in Chapter 2. This review first discusses the literature on Model-based Testing in general. It then continues with
a discussion on the literature of modelling languages, model transformations and toolsets that enable model transformations.

Chapter 3 provides an overview on the research methods that were applied for this dissertation. Where appropriate, alternative methods are discussed and evaluated.

Chapter 4 summarises the results that were generated by the research methods. A detailed listing of the results is provided in the Appendices of this dissertation. Chapter 4 completes with an analysis of the results and a validation on how far the research question had been answered.

Chapter 5 evaluates the research project work with respect to the objectives that were defined in Section 1.7. This chapter also suggests further work that results from this research project.
Chapter 2  Literature Review

The body of knowledge related to this research project covers a broad range of methodologies and concepts constituting the software testing discipline of Model-based Testing. This review of the body of knowledge describes these methodologies and concepts and shows how they relate to this research project.

The literature review starts with an introduction of the various concepts and methodologies of MBT and relates these concepts to this dissertation. This is followed by a discussion on the underlying concepts of the Intermediated Format 2.0 and the SDL-UML profile.

In order to facilitate the further discussion on SDL-UML and IF 2.0 modelling concepts, the following naming convention will be applied:

1. SDL-UML stereotypes will be written in italic letters and enclosed in brackets, e.g. \<< Interface >>.
2. UML Classes from the UML superstructure specification (OMG 2007b) will be written in italic letters, e.g. ReceiveSignalEvent.
3. IF 2.0 modelling concepts will be underlined, e.g. Process.

2.1 Concepts of Model-based Testing

The transformation of SDL-UML models to the Intermediated Format 2.0 is embedded with the software testing discipline of Model-based Testing. Utting and Legeard (2007) provide a comprehensive entry point to the field of MBT. They define MBT as the generation of test cases with expected output values from a behavioural model of a system under test (SUT). Utting and Legeard state that the MBT process can be abstracted and separated into the following five tasks:
1. A model of the system is created.
2. Abstract test cases are generated from this model.
3. Abstract test cases are concretized in order to make them executable.
4. Executable test cases are run against the SUT with verdicts being assigned.
5. The test results are analyzed.

Pretschner and Philips (2005) follow the same definition of MBT as Utting and Legeard. They elaborate this definition further by showing the relationship between a behavioural model and test cases that are derived from this behavioural model. According to Pretschner and Philipps a behavioural model represents the intended and observable behaviour of a SUT as specified by the user requirements. This observable behaviour consists of inputs that are being fed into the SUT and outputs that are being received from the SUT. Jerón (2004) defines sequences of observable behaviour as traces. Pretschner and Philipps follow the same definition of traces and state that they can be interpreted as test cases.

Pretschner and Philipps define models as mappings from the real world into a more abstract world. This level of abstraction needs to be considered when it comes to generation of test cases from a behavioural model. The reason for this is as follows. The definition of traces from a behavioural model results in a set of abstract test cases. These abstract test cases represent the same level of abstraction as the behavioural model itself (Pretschner and Philipps 2005). For that reason, abstract test cases are not concrete enough in order to run them against the SUT. It is therefore necessary to concretize abstract test cases in order to make them executable. This process is supported by technologies like the Test and Test Control Notation (TTCN-3), or the toolset and methodology provided by the
AGEDIS project. Baker et al. (2008) provide an overview on how the various parts of the TTCN-3 specification realise the transformation of abstract test cases to executable test cases. Another approach for the generation of executable test cases from abstract test cases is given by Hartmann and Nagin (2004) for the AGEDIS project.

An important aspect of MBT is that abstract test cases can be generated by abstract test case generators. An example of such an abstract test case generator is the TGV engine as described by Jard and Jerón (2005). They state that the TGV engine requires two types of input: a behavioural model and a specification of the traces that need to be tested. The specification of these traces is often referred to as test directives. These test directives can contain a number of parameters like system states and transitions that need to be covered.

The scope of this research project lies within the first two tasks of the MBT process as outlined by Utting and Legeard (2007). The transformation of SDL-UML to IF 2.0 can be allocated to the first task of system modelling. The verification of the research project hypothesis by comparison of manually and automatically generated abstract test cases can be allocated to the second task of abstract test case generation.

2.2 The Intermediated Format 2.0

It is the aim of this research project to investigate to what extent it is possible to transform the SDL-UML profile to the Intermediated Format 2.0. The Intermediated Format 2.0 was developed at VERIMAG (Graf et al. 2006b) together with a toolset that forms an architecture for IF 2.0 model validation and
exploration (Bozga et al. 2000, 2004b). The objective for this development had been to elaborate a toolset and intermediate language representation that would allow for simulation, verification, testing and debugging of real-time system models.

Figure 7 provides an overview on the IF 2.0 toolset architecture and provides examples on how IF 2.0 was utilised for commercial products like ObjectGeode, or research projects, like the AGEDIS and OMEGA project.
The IF 2.0 toolset is based on the following methodology (Bozga et al. 2004a). In a first step, a system model is transformed to the Intermediated Format 2.0. This IF 2.0 system model is then parsed by the IF 2.0 exploration engine into an abstract syntax tree (AST) that is represented as a collection of C++ objects. In a next step, the AST is transformed into a labelled transition system that can be utilised for system model exploration, simulation, or the generation of abstract test cases. Within the IF 2.0 toolset, the generation of abstract test cases is realised by the TGV engine.

Bozga et al. (2004a) provide examples of case studies where the IF 2.0 language and toolset were applied. These case studies had been conducted in the areas of telecommunication protocols, or embedded and distributed systems. Ober et al. (2006) describe an example for the application of the Intermediated Format 2.0 for the modelling of an embedded and distributed system. They provide a proof of concept for the transformation of an OMEGA UML 1.4 system model to the Intermediated Format 2.0. This OMEGA UML system model represented the Ariane-5 flight software that controls the flight of the multistage Ariane-5 rocket in an autonomous way.

Bozga et al. (1999a) state that the Intermediated Format 2.0 is general enough to cover subsets of concepts from other languages like UML 1.4 or SDL-92. This was proven in a number of research projects like the AGEDIS project, or the OMEGA project as outlined in Graf (2008). The aim of the OMEGA project was to provide a framework for the development of distributed real time systems (Information Society Technologies n.d). A deliverable of the OMEGA project was the OMEGA UML profile. This OMEGA profile is based on UML 1.4 and has
been transformed to IF 2.0 for the purpose of model verification and test case
generation. Graf et al. (2006b) show in their work how the OMEGA UML profile
had been mapped to the Intermediated Format 2.0.

Graf et al. state that the Intermediated Format 2.0 enables the definition of
distributed reactive systems. According to them, an IF 2.0 system consists of a
number of communicating processes that run in parallel. Each process has its own
process id (PID) and can contain a number of data variables. These data variables
can be either of a predefined type like Integer and Boolean, or of a more complex
abstract data type. Processes communicate either via asynchronous signals, shared
variables, or synchronous rendezvous. Asynchronous signals are exchanged on
signal routes and stored in process buffer queues.

Figure 8 shows an example of two IF 2.0 processes and how these processes can
communicate with each other.

Figure 8 Structure and communication between IF 2.0 processes (source: modified
from Graf et al. 2006b)
Graf et al. (2006b) describe the mapping of modelling concepts of the OMEGA profile (based on UML 1.4) to modelling concepts of the Intermediated Format 2.0. One important aspect of their work is that they showed how they had implemented the transformation of object-oriented concepts like inheritance to the Intermediated Format 2.0. For the case of inheritance this had been achieved in the following way. Graf et al. "flattened" inheritance by replicating all attributes from the IF 2.0 super-process type into the inheriting sub-process type. The work of Graf et al. is important for this research project, because it illustrates a workable solution for the transformation of an UML profile to IF 2.0.

One might argue that this workable solution could be reused for the transformation of the SDL-UML profile to IF 2.0. This is not the case for the following reasons. First, the transformation of the OMEGA profile to IF 2.0 was discussed on a general level and detailed mapping rules were omitted. Second, the transformation of the semantics of the OMEGA execution model to IF 2.0 was not discussed in the work of Graf et al. (2006b).

### 2.3 UML 2.0 and the SDL-UML profile

The UML 2.0 language specification contains two concepts that are important for the transformation of SDL-UML to IF 2.0 system models. These concepts are the UML 2.0 domain specific extension mechanism named "UML 2.0 Profiles" and "UML 2.0 modelling capabilities". This section illustrates how these two concepts relate to each other and how they relate to the transformation of SDL-UML to IF 2.0.
The UML 2.0 language specification is based on two publications of the Object Management Group (OMG). The first publication is the UML 2.0 infrastructure specification as outlined in OMG (2007a). This UML 2.0 infrastructure specification explains how UML 2.0 is built upon the Meta Object Facility 2.0 (OMG 2006). The second OMG publication is the UML 2.0 superstructure specification (OMG 2007b).

This specification defines UML 2.0 modelling concepts according to three logical areas. The first area is named "Part I - Structure" and contains all UML 2.0 modelling concepts that are related to the specification of system structure. The second area is called "Part II - Behavior" and contains all UML 2.0 modelling concepts that are related to the specification of system behaviour. The third area is named "Part III - Supplement" and contains all UML 2.0 modelling concepts for the customisation of UML 2.0 for various domains or platforms. UML 2.0 profiles are defined in this third area.
Next to UML 2.0 modelling capabilities, Figure 9 shows the second important UML 2.0 concept for this research project, i.e. UML profiles. These UML profiles provide a lightweight extension mechanism for the existing UML 2.0 metamodel. This is achieved by extending UML 2.0 metaclasses, or by applying constraints to these UML 2.0 metaclasses (OMG 2007b). UML profiles do not allow modifying or introducing new UML 2.0 metaclasses. An UML profile represents a specific UML package that contains stereotypes and constraints. These stereotypes can contain attributes and represent UML metaclass that have the ability to extend existing UML 2.0 metaclasses. Extending an UML 2.0 metaclass with a stereotype that contains attributes means that this UML 2.0 metaclass is also
extended by these attributes. So called tagged values assign values to these stereotype attributes.

The International Telecommunication Union (ITU) used this UML 2.0 domain extension mechanism by defining a SDL-UML profile as outlined in (ITU-T 2007). This SDL-UML profile maps SDL-2000 modelling concepts to UML 2.0 modelling concepts. The advantage of this SDL-UML profile is that it becomes possible to model systems according to SDL-2000 semantics with the support of existing UML tools.

Figure 10 provides an overview on how an UML 2.0 stereotype extends an UML 2.0 metaclass. An important aspect of Figure 10 is that it shows how elements of a stereotype affect the syntax and semantics of an UML metaclass. As shown in Figure 10, constraints of a stereotype limit the syntax of a metaclass. For example, this would mean that an UML metaclass can no longer own two forms of behavior like state machines or activities. Instead, the constraint can state that only state machines are allowed for the affected UML metaclass.

Figure 10 also shows that additional attributes can extend the syntax of an UML metaclass. All these extension and limitations of a UML metaclass cause a change in the semantics of a UML metaclass. For example in the SDL-UML profile specification, changes to the syntax of UML metaclasses are accounted for in the specification of additional semantic clauses.
Figure 10 Extension of UML metaclass by a stereotype

2.4 Syntax and semantics of the SDL-UML profile and IF 2.0

The previous chapter outlined the importance of syntax and semantics with respect to the application of the SDL-UML profile. Syntax and semantics also play an important role for the transformation of SDL-UML to IF 2.0. In a first step, this section provides an overview on the syntactical and semantically structure of modelling languages. This is followed by a discussion on the syntactic and semantic differences between SDL-UML and IF 2.0. In a final step, this chapter outlines what needs to be considered for the transformation of SDL-UML to IF 2.0 with respect to the syntactic and semantic differences of both languages.

A comprehensive overview on the structure of modelling languages is given by Harel and Rumpe (2004). They state that the structure of a modelling language consists of two parts: the syntax and semantics. The syntax consists of a set of
legal elements that represent the constructs of a language. This set of legal elements is also referred to as the abstract syntax of a modelling language. In the UML 2.0 superstructure specification, each modelling concept is referenced with its underlying abstract syntax. Mapping this abstract syntax to a graphical or textual notation results in the so-called concrete syntax.

The semantics explain what these syntactical constructs mean. It is important to note that a syntax element can have more than one meaning (Harel and Rumpe 2004). In such a case, the different meanings of a syntax element are grouped in a semantic domain. This is further elaborated in the UML 2.0 superstructure specification (OMG 2007b) by showing that a modelling concept can be related to static and dynamic semantics. The difference between these two kinds of semantics is defined in the UML 2.0 superstructure specification. The static semantics describes what the syntax elements of a modelling concept mean. The dynamic semantics describes, where appropriate, the behaviour of these concepts.
Figure 11 Syntax and semantics in the SDL-UML to IF 2.0 transformation process

In the next section I will discuss how to convert the graphical syntax of SDL-UML to the textual syntax of the Intermediated Format 2.0.

2.5 Model transformations

The transformation of SDL-UML to IF 2.0 requires a well-defined transformation methodology. In order to decide which transformation methodology to choose for this research project, the following needs to be considered.

The transformation of SDL-UML to IF 2.0 represents the transformation of a diagrammatic language to a textual language. According to Czarnecki and Helsen (2003) this transformation from SDL-UML to IF 2.0 can be classified as a model-to-code transformation. Figure 12 provides an overview on the classification of model transformation approaches as outlined by Czarnecki and Helsen (2003).
Model-to-code transformations can be classified to visitor-based and template-based transformation approaches (Czarnecki and Helsen 2003) as shown in Figure 12. Czarnecki and Helsen state that a visitor-based approach provides a mechanism to traverse the internal structure of a model and then write text to a text stream. In contrast to this, template-based transformation approaches contain templates that exist of text of the target language and placeholders. These placeholders are expressions for selecting values from the source model (OMG 2008a).

Model-to-code transformations are realised by a number of transformation languages and tools (Figure 12). Examples of such tools are the openArchitectureWare platform that implements the template-based transformation language Xpand (Klatt 2007), or the Eclipse based MOFScript tool that implements the MOF Model to Text Transformation Language (OMG 2008a).
Both tools follow a template-based model-to-code transformation approach.

The major difference between the MOF Model to Text Transformation Language and other languages like Xpand is that it is the only model-to-code transformation language that is based on an official standard, i.e. OMG (2008a). For this reason, it will be used for the SDL-UML profile to IF 2.0 transformation. The MOF Model to Text Transformation Language is supported by the Eclipse based MOFScript tool. The MOFScript tool integrates into the Eclipse architecture as shown in Figure 13.

Figure 13 MOFScript tool and the Eclipse Modelling Framework

The MOFScript tool is a deliverable of the Eclipse Generative Modelling Technologies (GMT) project which provides a set of modelling tools that support model-driven system development. The MOFScript tool is based on the Eclipse Modeling Framework (EMF). The EMF is a framework and code generation facility that supports the development of Eclipse tools that are based on models (Carlson 2005). Next to the MOFScript tool, Figure 13 shows the Eclipse UML
2.0 component. This UML 2.0 component is an EMF based implementation of the UML 2.0 specification as outlined in OMG (2007 a, b). The Eclipse UML 2.0 component is of importance for this research project because it enables the specification of SDL-UML system models that can be stored in the Eclipse EMF environment. These SDL-UML models are then used by the MOFScript tool for the transformation of SDL-UML models to IF 2.0.

Oldevik (2006) states that the MOFScript tool contains two logical sets of components: tool components and service components as shown in Figure 14. The service components enable parsing, model syntax checking, and the execution of model transformations. The tool components provide functionality that enables the user to edit MOFScript files and to interact with MOFScript service components.

![Figure 14 Architecture of MOFScript tool (modified from source: Oldevik 2006)](image)

The functionality of the MOFScript tool can be described as follows:
1. The Text Editor allows the user to edit transformation files, to invoke the parser or the semantic checker, and to invoke the transformation execution engine.

2. The Outline and Preference Viewer, as well as the Preference Manager support the user in writing and executing transformations.

3. The Result Manager provides the functionality of storing transformation results.

Once the user starts the transformation process, the service components provide the following functionalities:

1. The Parser and Lexer check for the correct syntax of the content of the transformation file. Once this has been completed successfully, the Parser and Lexer calls the Model Manager in order retrieve the system model and its metamodel as specified in the Text Editor.

2. The Model Manager populates the MOFScript metamodel with elements from the SDL-UML system model.

3. The Semantic Checker ensures the correctness of the transformation rules. The execution of the service components is then directed by the Execution Engine.

2.6 The AGEDIS-TGV engine

As discussed in Section 1.5, the AGEDIS-TGV is one software component of the AGEDIS toolset. It is important to note that the AGEDIS-TGV engine differs from the TGV engine originally developed by VERIMAG (Jard and Jerón 2005). The main difference between these two engines is that the AGEDIS-TGV engine outputs abstract test cases only in one format. Nagin and Kirshin (2003) describe this format in more detail. According to them, the abstract test case format of the AGEDIS-TGV engine is based on a XML schema developed by the AGEDIS project team. In order to distinguish the two TGV engines, the TGV engine originally developed at VERMIAG shall be further referred to as the VERIMAG-TGV engine.
The AGEDIS-TGV engine is one of the most important software components of the AGEDIS toolset. Within the scope of this research project, the AGEDIS-TGV engine will be used for automatically generating abstract test cases from an IF 2.0 system model. A comprehensive overview on the underlying theory of the AGEDIS-TGV engine is given by Jard and Jerón (2005). They state that the TGV engine is based on the theory of conformance testing of non-deterministic reactive systems. In this respect, reactive means that the SUT responds to stimuli and non-deterministic means that one stimulus might result in different system reactions. Jard and Jerón define conformance testing as a special form of functional black-box testing. Within a conformance testing approach, the actual behaviour of the SUT is tested against a behavioural model. The interaction with the SUT is achieved via inputs being sent to the SUT and outputs being received from the SUT.

Jerón (2004) states that the AGEDIS-TGV engine requires two types of input: a specification of the behavioural model and test directives (Figure 15). The AGEDIS-TGV engine requires the behavioural model to be specified in one of the three modelling languages: UML 1.4, SDL-92, or the Intermediated Format 2.0. Jerón (2004) provides an overview on the three elements a test directive can contain: a test purpose, data constraints and test coverage directives. Test purposes are modelled as automata and describe the expected traces of the system behaviour. Data constraints provide the means to specify that certain states along the traces should or should not be covered. Test coverage directives allow the user to specify that all values of a specific attribute must be covered by the generated abstract test cases.
Figure 15 illustrates the inputs and the outputs of the AGEDIS-TGV engine, as well as the various processing steps that the AGEDIS-TGV engine performs for generating abstract test cases. According to Jard and Jerón (2005), the AGEDIS-TGV engine computes in a first step the synchronous product of the behavioural model and the test purpose. This results in a marking of the system behaviour that is either accepted or refused by the test purpose. In a next step, the AGEDIS-TGV engine computes the visible behaviour from the synchronous product. Then, the AGEDIS-TGV engine builds abstract test cases by selecting the visible behaviour as specified in the test directives. In a final step, controllability conflicts within these abstract test cases are resolved.

Figure 15 The AGEDIS-TGV engine (modified from source: Jerón 2004)
2.7 Research question

It is the aim of this research project to investigate to what extent it is possible to transform SDL-UML stereotypes to modelling concepts of the Intermediated Format 2.0 for the purpose of automatic test case generation.

2.8 Summary

The literature review revealed that a number of research projects implemented transformations of system models to the Intermediated Format 2.0. Examples for transformations of UML 1.4 profiles to IF 2.0 are the OMEGA project as described in Graf et al. (2006b), or the AGEDIS project as outlined by Hartmann and Nagin (2004). Another example for a SDL-92 to IF 2.0 transformation is given by Bozga et al. (1999b).

The evolution of SDL-92 to SDL-2000 and the evolution of UML 1.4 to UML 2.0 contained major revisions for both languages. Consequently, the existing transformations from SDL-92 and UML 1.4 to IF 2.0 can not be reused for the transformation of the SDL-UML profile to the Intermediated Format 2.0. Nevertheless, the literature review showed that the AGEDIS toolset and the MOFScript tool could be flexible enough to enable SDL-UML to IF 2.0 transformations.
Chapter 3 Research Methods

This chapter describes the research methods that were applied in order to meet the objectives of this dissertation. Prior to discussing each research method in detail, a brief overview shows how they relate to each other.

The first research method analysed the UML 2.0 modelling concepts that had been used for the specification of the SDL-UML profile and its respective stereotypes. This research method generated categories for the grouping of SDL-UML and IF 2.0 modelling concepts.

These categorised SDL-UML modelling concepts were then used for the modelling of a car parking ticket machine system. The behaviour of the SDL-UML ticket machine model was similar to the behaviour of an IF 2.0 ticket machine model that was provided by the AGEDIS toolset (Hartmann and Nagin 2004). This similarity was of importance for the verification of the hypothesis of this dissertation.

The SDL-UML modelling concepts that had been used for the modelling of the ticket machine provided the input for a comparative analysis between modelling concepts of the SDL-UML profile and the Intermediated Format 2.0. This comparative analysis resulted in a hypothesis to what extent it was possible to transform SDL-UML modelling concepts to IF 2.0 for the purpose of automatic test case generation.

As a next step, MOFScript transformation templates were specified for the SDL-UML modelling concepts that had been used for the modelling of the ticket
machine system. These transformation templates transformed SDL-UML stereotypes to IF 2.0 modelling concepts. All these transformations were based on the assumption that IF 2.0 processes use their process identifiers for signal exchange and not explicitly defined signalroutes.

The verification of the SDL-UML stereotype transformation hypotheses was then performed in the following way. Each MOFScript generated IF 2.0 modelling concept was compared to an equivalent IF 2.0 modelling concept in the AGEDIS IF 2.0 ticket machine example. If a SDL-UML stereotype could be transformed to an IF 2.0 modelling concept that was used in the AGEDIS IF 2.0 ticket machine example, then the hypothesis was considered as being verified.

3.1 Analysis of SDL-UML modelling concepts

The analysis of SDL-UML modelling concepts had two objectives. The first objective was to analyse the static and dynamic semantics of the SDL-UML modelling concepts. This objective was necessary for the following reason. In order to understand how the SDL-UML stereotypes extend the underlying UML 2.0 modelling concepts, the static and dynamic semantics of these UML 2.0 modelling concepts had to be first analysed. The second objective was to identify categories for the grouping of the identified UML 2.0 modelling concepts. This grouping according to behavioural and structural categories facilitated the modelling of the ticket machine system.

The UML 2.0 modelling concepts that had been used for the specification of the SDL-UML profile were identified in the following way. Each SDL-UML stereotype definition contains a clause that references the UML 2.0 modelling
concepts that had been used for the specification of this stereotype. For each of
these UML 2.0 modelling concepts, the static and dynamic semantics were
analysed and the columns in Table 1 completed. This method resulted in an
overview of all the UML 2.0 modelling capabilities and concepts that had been
used for the specification of the SDL-UML stereotypes.

It is also important to note that the static and dynamic semantics were not
reproduced in Table 1, but referenced with the column named "Clause in UML
2.0 superstructure specification".

|----------------------------------------------------------|----------------------------------------------------------|---------------------------------------------|---------------------------------------------|-----------------------------------|--------------------------------------|----------------------------------|

Table 1 Overview on SDL-UML modelling capabilities and underlying concepts

3.2 Specification of a SDL-UML system model

The categorised SDL-UML modelling concepts from the previous method were
then used for the modelling of a reference system that could be used for the
following research methods. This reference system had to fulfil the following
requirements:

1. The reference system had not to be too complex.
2. The reference system had to own an observable behaviour that would allow for the generation of abstract test cases.
3. The reference system had to own a structure that would allow for the modelling of "structural" SDL-UML modelling stereotypes like <<activeClass>>, <<Port>>, <<Interface>>, etc.
A search for a reference system that would fulfil these criteria revealed a number of systems that were in most cases either too complex to be modelled within the given time budget for this dissertation. An example for a system where this was the case is outlined in SINTEF (n.d.). A system that fulfilled the above listed requirements was provided by the AGEDIS toolset. This system represented a ticket machine and provided the following functionality as outlined in the AGEDIS IF 2.0 ticket machine example as shown in Appendix F:

1. Inserting coins into the ticket machine.
2. Removing coins from the coin tray.
3. Removing a ticket from the ticket tray.
4. Requesting a ticket.
5. Removing coins from the coin buffer.

The SDL-UML ticket machine model was specified with the UML modelling tool MagicDraw v16.6 (No Magic, Inc. 2009) and a SDL-UML profile specification that was provided by the Fraunhofer ESK Institute (Institute ESK 2009). It became soon obvious from the modelling of the ticket machine system that not all SDL-UML stereotypes could be used for this ticket machine model. This fact helped to reduce the number of existing SDL-UML stereotypes to a reasonable number that would allow accomplishing the following research methods within the available time budget.

An important aspect for the SDL-UML ticket machine model was the question on how to model the communication between UML classes of stereotype <<activeClass>>. Valid options of communication between SDL-UML stereotypes of kind <<activeClass>> were asynchronous communication via
signal exchange, or synchronous communication via operation calls. For this SDL-UML ticket machine model the design decision was made to model the communication between ticket machine classes as asynchronous communication via signal exchange.

3.3 Comparative analysis between SDL-UML and IF 2.0 modelling concepts

The input to the comparative analysis was the SDL-UML stereotypes that had been used for the modelling of the ticket machine system. The comparative analysis had the objective to identify to what extent these SDL-UML stereotypes could be transformed to IF 2.0 modelling concepts for the purpose of automatic test case generation.

The comparative analysis started with a grouping of SDL-UML and IF 2.0 modelling concepts according to the categories specified in Table 1. For each IF 2.0 modelling concept, the static and dynamic semantics were studied. These static and dynamic semantics were then compared to

1. the static and dynamic semantics of the UML 2.0 modelling concepts listed in Table 1,

2. and where appropriate, to the modification of these static and dynamic semantics through additional semantic clauses, attributes and constraints in the SDL-UML stereotype definitions. This extension mechanism was discussed in Chapter 2.3 and illustrated in Figure 11.

Based on the result of this comparison, a hypothesis was formulated to what extent an SDL-UML stereotype could be transformed to an IF 2.0 modelling concept. The following needs to be considered with respect to the comparison
between the static and dynamic semantics of both modelling languages. The static and dynamic semantics of the UML 2.0 language specification are formulated in written human language. This type of specification can not be considered as being very formal and leaves room for ambiguity. This means the following for this comparative analysis. The interpretation of the static and dynamic semantics of the SDL-UML stereotypes depends on how the person performing this comparative analysis interprets these semantics. Verification of these interpretations can only be achieved at the stage of the final verification method.

The results of the comparative analysis were written into a table that had a structure as outlined in Table 2. The entire set of results is shown in Appendix C.

<table>
<thead>
<tr>
<th>Name of stereotype (ITU-T 2007)</th>
<th>SDL-UML stereotype element identifier</th>
<th>SDL-UML stereotype element</th>
<th>SDL-UML stereotype element description</th>
<th>Comment</th>
<th>IF 2.0 modelling concept description</th>
<th>IF 2.0 Element BNF Specification</th>
<th>Hypothesis on transformability for automatic test case generation</th>
</tr>
</thead>
</table>

Table 2 Overview on structure of comparative analysis results table.

The comparative analysis completed with a number of hypotheses to what extent SDL-UML stereotypes could be transformed to modelling concepts of the Intermediated Format 2.0 for the purpose of automatic test case generation. The following research methods were executed in order to verify these hypotheses.

### 3.4 Specification of transformation templates

The specification of transformation templates followed a template based transformation methodology as outlined in Chapter 2.5. The inputs to this method were mapping rules that had been based on the transformation hypotheses in
Table 2. The specification of transformation templates was achieved in the following way.

The MOFScript tool with version 1.3.3 was used for the specification of the transformation templates. These templates contained text of the IF 2.0 target system and placeholders in form of queries to the SDL-UML system model. The text in these transformation templates represented concrete syntactical constructs of the IF 2.0 that were completed with values from the queries to the SDL-UML system model. The transformation template is listed in Appendix H.

3.5 MOFScript transformation

The transformation process was realised with the Eclipse MOFScript tool that required the following inputs as illustrated in Figure 16. The first input was an XMI (OMG 2007c) specification of the SDL-UML model as outlined in Appendix D. This SDL-UML model was imported into the Eclipse EMF with a plug-in that had provided by the Fraunhofer ESK Institute (Institute ESK 2009). The second input was the UML 2.0 Metamodel of the SDL-UML system model. The third input to the MOFScript tool was the transformation template as specified in Appendix H.

With all these three types of input in place, the MOFScript tool executed the transformation process and output an IF 2.0 system model as shown in Appendix G. This IF 2.0 system model was used as input to the AGEDIS TGV engine for the purpose of automatic abstract test case generation.
3.6 Alternative transformation methods

Two alternative transformation methods were considered prior to the choice for the MOFScript tool based transformation methods. These two alternative transformation methods were the Xpand language as part of the openArchitectureWare platform and the XSLT language. This section explains how these two alternative transformation methods would have implemented the transformation from SDL-UML to IF 2.0.

The first alternative transformation method would have been based on the XSL Transformations (XSLT) language. This XSLT language was specified by the World Wide Web Consortium (W3C) and is widely used as a XML-based language for the transformation of XML documents (Tidwell 2001). The XSLT technology requires one or more XML documents that are transformed by a XSLT...
processor according to XSLT stylesheet specifications. The alternative method for the transformation of SDL-UML to IF 2.0 would have been implemented as illustrated in Figure 17.

Figure 17 XSLT based transformation method

For this XSLT stylesheet based method, the XSLT processor requires two types of input. The first type of input is a SDL-UML system model in XMI format. This XMI based SDL-UML system model represents a XML document that can be further processed by the XSLT processor according to XSLT stylesheets specifications.

These XSLT stylesheets represent the second type of input to the XSLT processor. They contain the SDL-UML to IF 2.0 transformation rules that have been identified during the comparative analysis. The XSLT processor executes the XSLT stylesheets on the XMI system model and outputs an IF 2.0 system specification.
This XSLT based transformation method will not be considered as an alternative to the MOFScript based transformation. The reasons for this are as follows. The first reason is that the XSLT transformation method is not based on a model-to-text transformation standard. In comparison to this, the MOFScript tool is based on the MOF Model to Text Transformation Language (OMG 2008b). Therefore, the MOFScript tool transformations are based on an industry-wide acknowledged transformation methodology. The second reason is that the XSLT based transformation method does not support the semantic checking of the source model that is transformed. In comparison to this, the MOFScript tool supports this semantic checking functionality by using the UML 2.0 metamodel of the SDL-UML profile.

The second alternative transformation method would have been based on the language Xpand and its implementation in the openArchitectureWare platform. Similar to MOFScript, Xpand also enables the specification of transformation rules based on the metamodel of the input model. In addition, Xpand and MOFScript use the infrastructure of the Eclipse Modelling Framework (EMF). Klatt (2007) states that Xpand is not based on the MOF Model to Text Transformation Language as defined in OMG (2008b). It had been one objective of this research to deploy, if possible, research methods that were based on acknowledged standards. The reason for this was to make the research methods of this dissertation comparable to other research performed in this area. For this reason the decision was made to use the MOFScript tool for the transformations of SDL-UML to IF 2.0.
3.7 Verification of transformation hypotheses

Two alternative methods were considered for the verification of the transformation hypotheses. Each of the two verification methods shall be described in more detail as follows.

3.7.1 Verification method 1

The first verification method consisted of a comparison of manually and automatically generated abstract test cases. In order to make them comparable, both sets of abstract test cases were based on the same traces of system behaviour. These traces were specified from the behavioural part of the SDL-UML ticket machine model.

Manually generated abstract test cases were specified with the AGEDIS XML schema as discussed in Section 1.2. The specification of automatically generated abstract test cases followed the AGEDIS-TGV engine methodology as outlined in Section 1.5. For this verification method, the AGEDIS-TGV engine methodology required two kinds of input. The first input was an IF 2.0 specification of the SDL-UML ticket machine model. The second input was the same specification of behavioural traces in form of test directives, as it was the case for the manually generated abstract test cases. The AGEDIS-TGV engine then outputted abstract test cases that were specified with the AGEDIS XML schema as discussed in Section 1.2.

The method of abstract test case generation completed with a set of manually and automatically generated abstract test cases. This set provided the input to the
verification method of the transformation hypothesis in the following way.

First, the manually and automatically generated abstract test cases were grouped according to the traces they covered. Second, for each group of manually and automatically generated abstract test cases the following criteria had been compared: coverage of states, coverage of transitions, inputs to the SUT and outputs from the SUT.

If the manually and automatically generated abstract test cases were equal in terms of the comparison criteria, then the transformation hypothesis was considered as being verified.

3.7.2 Verification method 2

The second verification method was based on three types of input. The first input were the IF 2.0 modelling concepts that had been generated by the MOFScript transformation template (Appendix H). The second input was the AGEDIS IF 2.0 ticket machine example (Appendix F). The third input was an automatically generated abstract test case (Appendix I) that covered all functions and processes of the AGEDIS IF 2.0 ticket machine example. This abstract test case was generated by the AGEDIS TGV-engine according to the following test directives shown in Figure 18.

directive "testPurposeAllOperations";

state tpInsertCoinStart #start ;

match < input IF_call_TicketMachine_insertCoin() > ;

nextstate State;

endstate;
The IF 2.0 modelling concepts that had been generated by the MOFScript tool were then compared to the IF 2.0 modelling concepts in the AGEDIS IF 2.0 ticket machine example (Appendix F). If a MOFScript generated IF 2.0 modelling concept had an equivalent in the AGEDIS IF 2.0 ticket machine example, then the transformation hypothesis for the respective SDL-UML stereotype was considered as being verified. For this verification method it was assumed that the SDL-UML ticket machine example had been a true representation of the AGEDIS IF 2.0 ticket machine specification (Appendix F).
For example, the MOFScript transformation template would transform a SDL-UML stereotype of \texttt{<<activeClass>>} to an IF 2.0 modelling concept of \texttt{Process}. This \texttt{Process} is also being used in the AGEDIS IF 2.0 ticket machine specification. From this specification, abstract test cases can be generated that cover this IF 2.0 modelling concept of \texttt{Process}. In such a case, the hypothesis that the SDL-UML stereotype of \texttt{<<activeClass>>} could be transformed to an IF 2.0 modelling concept of \texttt{Process} for the purpose of automatic test case generation would be considered as being verified.
Chapter 4 Results

This chapter describes the results that were generated by the research methods of this dissertation.

4.1 Results from the analysis of SDL-UML stereotypes

The analysis of SDL-UML stereotypes produced two sets of results. The first result was an overview on the UML 2.0 modelling concepts that were used for the specification of each SDL-UML stereotypes. This result was of importance for the specification of the SDL-UML ticket machine model for the following reason. The structured overview on the SDL-UML stereotypes and their respective UML 2.0 modelling concepts facilitated the modelling of various structural and behavioural features in the SDL-UML ticket machine model. The overview on the various SDL-UML stereotypes and their underlying UML 2.0 modelling concepts is provided in Appendix B.

The second result of this analysis was categories for the grouping of UML 2.0 modelling concepts that had been used for the specification of SDL-UML stereotypes. These categories are represented as UML 2.0 modelling capabilities in Table 3. This set of categories was of importance for the comparative analysis in Section 4.3 because they allowed the structuring of SDL-UML and IF 2.0 modelling concepts. This ensured that like with like was being compared. The entire grouping of UML 2.0 modelling concepts according to these categories is shown in Appendix B.
<table>
<thead>
<tr>
<th>Part in UML 2.0 superstructure specification (OMG 2007b)</th>
<th>Category - UML 2.0 modelling capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part I – Structure</td>
<td>Classes</td>
</tr>
<tr>
<td>Part I – Structure</td>
<td>Composite Structures</td>
</tr>
<tr>
<td>Part II – Behavior</td>
<td>Actions</td>
</tr>
<tr>
<td>Part II – Behavior</td>
<td>Activities</td>
</tr>
<tr>
<td>Part II – Behavior</td>
<td>Common Behaviors</td>
</tr>
<tr>
<td>Part II – Behavior</td>
<td>State Machines</td>
</tr>
<tr>
<td>Part III – Supplement</td>
<td>Auxiliary Constructs</td>
</tr>
</tbody>
</table>

Table 3 List of UML 2.0 modelling capabilities used for the specification of the SDL-UML profile

4.2 Results from the specification of a SDL-UML ticket machine model

The SDL-UML ticket machine model was specified in two parts. The first part covered the structural aspects and the second part covered the behavioural aspects of the ticket machine model.

For the specification of the structural part, two types of UML diagrams were used. The first diagram type was an UML Composite Structure Diagram as shown in Figure 19. The reason for choosing this diagram type was as follows.

With a UML Composite Structure Diagram it was possible to show the ticket machine system boundaries and system interaction points. These system interaction points are represented in SDL-UML with the stereotype `<Port>`.

As discussed in Section 1.1, Model-based Testing includes the sending of signals to the system under test (SUT) and the receiving of signals from the system under test. The UML Composite Structure Diagram in Figure 19 clearly shows SDL-
UML stereotypes of type «Port», through which signals can be sent to and received from the SDL-UML ticket machine system.

The method of SDL-UML ticket machine modelling not only produced a UML Composite Structure Diagram, but also detected an ambiguity in the SDL-UML profile specification (ITU-T 2007). There, in the semantic clause of the stereotype specification of «activeClass» the following is stated. If the isConcurrent attribute of the stereotype «activeClass» is equal to true, then this «activeClass» represents an SDL agent type of kind block. Otherwise, if the isConcurrent attribute is equal to false then the «activeClass» represents an SDL agent type of kind process. An «activeClass» for the agent kind system is missing in the SDL-UML profile specification, although this agent kind is specified in the SDL-2000 specification (ITU-T 2002).

This ambiguity had consequences for the modelling of the system class in the SDL-UML Composite Structure Diagram and the creation of MOFScript transformation templates. An «activeClass» that represented an agent type of kind system was therefore modelled as the outermost «activeClass» with the attribute isConcurrent being equal to true. This missing specification of the agent type of kind system also had to be considered for the design of the MOFScript transformation templates. A transformation rule for selecting the «activeClass» that represented the ticket machine system had to search for the outermost «activeClass» in the SDL-UML ticket machine model.
The second diagram type that was used for the specification of the structural part of the SDL-UML ticket machine model was an UML Class Diagram as shown in Figure 20. This UML Class Diagram illustrated the number of interfaces that were realised or used by the ports shown in Figure 20. These interfaces specify the number and type of signals that a SDL-UML <<activeClass>> Class provides or requires from its environment.
UML State Machine and Activity Diagrams were used for the specification of the behavioural part of the SDL-UML ticket machine example. With respect to the behavioural modelling the following has to be noted. Soon after the start of the behavioural modelling the following became obvious. Although the SDL-UML ticket machine model exhibits behaviour of limited complexity, the modelling effort for the entire ticket machine would have exceeded the available time budget for this dissertation. Therefore, the method of behavioural modelling had to be adjusted as follows.
For each `<activeClass>` within the ticket machine example a `<StateMachine>` was modelled. The adjustment of the behavioural modelling resulted in the fact that only the `<StateMachine>` of `<activeClass>` `TicketTray` as shown in Figure 21 also contained a behaviour `<Activity>` `returnFalse` for the transition that is triggered by the `ReceiveSignalEvent` with name `EmptyTicketTray`.

![StateMachine of TicketTray](image)

**Figure 21** `<StateMachine>` of `<activeClass>` `TicketTray`

The `<Activity>` `returnFalse` is specified as shown in Figure 22. This activity models the following behaviour. If the user of the SDL-UML ticket machine wants to empty the ticket tray, although there is no ticket available, the ticket
tray returns a signal to the ticket machine stating that there is no ticket for retrieval. This behaviour is achieved via the <<SendSignalAction>> in Figure 22 that assigns the value false to the Boolean attribute of the <<Signal>> EmptyTicketTray.

![Figure 22](image)

Figure 22  <<Activity>> returnFalse for ReceiveSignalEvent EmptyTicketTray

The SDL-UML modelling of the ticket machine system completed with a number of structural and behavioural diagrams as shown in this section and in Appendix D. The SDL-UML stereotypes that were used for the design of the SDL-UML ticket machine model provided the input for the following comparative analysis.

4.3 **Comparative analysis between SDL-UML and IF 2.0 modelling concepts**

Prior to the start of the comparative analysis, the SDL-UML stereotypes that were used for the SDL-UML ticket machine model and IF 2.0 modelling concepts were grouped together. This grouping was performed according to the categories
specified as a result of the analysis in Section 4.1. The grouping of the SDL-UML stereotypes and IF 2.0 modelling concepts resulted in the contents of Table 4.

<table>
<thead>
<tr>
<th>Part in UML 2.0 superstructure specification (OMG 2007b)</th>
<th>Category - UML 2.0 modelling capability</th>
<th>SDL-UML Stereotype from SDL-UML ticket machine model</th>
<th>IF 2.0 modelling concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part I - Structure</td>
<td>Classes</td>
<td>&lt;&lt;Package&gt;&gt; &lt;&lt;PrimitiveType&gt;&gt;</td>
<td>Not defined Predefined basic data type</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>Composite Structures</td>
<td>&lt;&lt;activeClass&gt;&gt; &lt;&lt;Port&gt;&gt; &lt;&lt;Interface&gt;&gt; &lt;&lt;Connector&gt;&gt; &lt;&lt;Property&gt;&gt;</td>
<td>Process/System Not defined Not defined Not defined Signalroute Process</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>Actions</td>
<td>&lt;&lt;sendSignalAction&gt;&gt;</td>
<td>output</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>Activities</td>
<td>&lt;&lt;Activity&gt;&gt; &lt;&lt;SequenceNode&gt;&gt; &lt;&lt;ActivityFinalNode&gt;&gt;</td>
<td>Action Not defined Not defined</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>Common Behaviors</td>
<td>&lt;&lt;Signal&gt;&gt;</td>
<td>Signal</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>State Machines</td>
<td>&lt;&lt;StateMachine&gt;&gt; &lt;&lt;State&gt;&gt; &lt;&lt;PseudoState&gt;&gt; &lt;&lt;Region&gt;&gt; &lt;&lt;Transition&gt;&gt;</td>
<td>Process States State #start, Endstate State #unstable Not defined Transition</td>
</tr>
<tr>
<td>Part III - Supplement</td>
<td>Auxiliary Constructs</td>
<td>Not used in SDL-UML model</td>
<td>Not defined</td>
</tr>
</tbody>
</table>

Table 4 Grouping of SDL-UML stereotypes and IF 2.0 modelling concepts

This categorisation of SDL-UML stereotypes and IF 2.0 modelling concepts produced two sets of results. The first set of results was pairs of SDL-UML stereotypes and IF 2.0 modelling concepts that showed similarities in terms of the categories in Table 4. The second set of result was SDL-UML stereotypes that had no equivalent in the IF 2.0 language specification.

Although some SDL-UML stereotypes in Table 4 had no equivalent in the IF 2.0 language specification, these SDL-UML stereotypes were still considered as input to the comparative analysis. The reason for this is as follows. As discussed in Section 2.2, Graf et al. (2006b) showed how UML modelling concepts that did not exist in the IF 2.0 language specification could be “flattened” within an IF 2.0
system specification. The term “flatten” refers to the activity of substituting one UML modelling concept with a number of IF 2.0 modelling concepts. Within a given context, this number of IF 2.0 modelling concepts would then have a semantics that is similar to the semantics of the substituted UML modelling concept. This aspect was considered for the following comparative analysis.

The comparative analysis started with those SDL-UML stereotypes that had an equivalent in the IF 2.0 language specification as shown in Table 4. For each of these pairs a transformation hypothesis was formulated. This transformation hypothesis was translated into one or more transformation rules during the specification of MOFScript transformation templates.

The comparative analysis then continued with those SDL-UML stereotypes that had no equivalent IF 2.0 modelling concept. Where appropriate, also for these SDL-UML stereotypes a transformation hypothesis was formulated.

4.3.1 Pair - SDL-UML: <<activeClass>> IF 2.0: System/Process, SDL-UML: <<Property>> IF 2.0: Process

In SDL-UML it is possible to define an <<activeClass>> Class as a property of an encapsulating Classifier of another <<activeClass>> Class. Therefore, the two pairs of modelling concepts, SDL-UML: <<activeClass>> IF 2.0: System/Process on the one side and SDL-UML: <<Property>> IF 2.0: Process on the other side, were analysed together.

For these two pairs of modelling concepts, the comparative analysis covered two questions. The first question was to what extent the execution model of SDL-
UML could be transformed to the execution model of the Intermediated Format 2.0. The second question was to what extent the composite structure of a SDL-UML model could be resolved to a “flat” structure of an IF 2.0 system specification.

With respect to the execution model of both modelling languages, the following had to be considered. Instances of \texttt{<<activeClass>>} of kind \texttt{Process} execute interleaved when they are contained within an \texttt{<<activeClass>>} of kind \texttt{Block}. In comparison to this, the IF 2.0 execution model only defines concurrently executing instances of \texttt{Process} (Bozga et al. 2004a). Therefore, instances of \texttt{<<activeClass>>} of kind \texttt{Process} that are contained within an \texttt{<<activeClass>>} of kind \texttt{Block} cannot be transformed to instances of IF 2.0 \texttt{Process}.

This is not the case for instances of \texttt{<<activeClass>>} of kind \texttt{Block} that are contained within an \texttt{<<activeClass>>} of kind \texttt{Block}. The reason for this is that instances of \texttt{<<activeClass>>} of kind \texttt{Block} execute concurrently. Therefore the hypothesis was formulated that \texttt{<<activeClass>>} of kind \texttt{Block} could be transformed to IF 2.0 \texttt{Process} with respect to the execution model of both modelling languages.

For the transformation of the composite structure of the SDL-UML ticket machine model the following had to be considered. In this model, the encapsulated Classifiers like \texttt{<<activeClass>> TicketTray} were specified as \texttt{<<Property>>} of the encapsulating \texttt{<<activeClass>> TicketMachine}. This way of defining a composite structure as outlined on OMG (2007b, Clause 9) is an important feature of the SDL-UML profile for the modelling of systems with underlying SDL.
The comparative analysis revealed that the concept of composite structures did not exist in the IF 2.0 language specification. As specified in Bozga and Olvovsky (2002) a Process must not contain another Process. It was therefore part of the comparative analysis to investigate how a composite UML structure can be transformed to an IF 2.0 system specification for the purpose of automatic test case generation. This type of transformation involved the “flattening” of a composite UML structure as discussed in Section 4.3. This meant for the context of the SDL-UML ticket machine model that each <<activeClass>> Class could be transformed into the “flat” structure of a number of instances of Process within a System specification. The major difference between the flat and composite structure was that the enclosed instances of <<activeClass>> Class within <<activeClass>> TicketMachine became visible at one System level.

The following transformation hypotheses were formulated, with respect to the previous discussion of the execution model of both modelling languages and the composite structure of SDL-UML.

Hypothesis 1: an outermost <<activeClass>> can be transformed to a System for the purpose of automatic test case generation.

Hypothesis 2: an <<activeClass>> <<Property>> Class of kind Block within the enclosing <<activeClass>> TicketMachine can be transformed to a Process for the purpose of automatic test case generation.
4.3.2 Pair – SDL-UML: <<Connector>> IF 2.0: Signalroute

The comparative analysis between the SDL-UML stereotype <<Connector>> and the IF 2.0 modelling concept Signalroute produced the following results.

Both modelling concepts share the similarity that they enable the communication either between instances of <<activeClass>> with respect to SDL-UML, or instances of Process with respect to IF 2.0. This is achieved for both languages via the exchange of signals. The major differences between <<Connector>> and Signalroute are as follows.

As shown in Bozga and Olvovsky (2002), Signalroute requires as part of its specification the definition of Signals that are transported via Signalroute. Figure 23 illustrates the specification of a Signalroute sig1(1) from process TicketMachine to TicketTray with signal EmptyTicketTray.

```java
signal EmptyTicketTray(boolean);
signalroute sig1(1) from TicketMachine to TicketTray with EmptyTicketTray;
```

Figure 23 Definition of an IF 2.0 Signalroute

The UML superstructure specification (OMG 2007b, Clause 9.3.6) and SDL-UML profile specification (ITU-T 2007) show that the specification of <<Connector>> does not include the specification of <<Signal>>. Instead, the UML superstructure specification defines the UML Class InformationFlow that can be linked to <<Connector>> and <<Signal>>. The UML Class InformationFlow allows for the linking of messages via the association realizingMessage. This is of importance when it comes to the transformation from
<<Connector>> to Signalroute. This transformation always has to consider the associations of InformationFlow to <<Connector>> and <<Signal>>.

Another difference between <<Connector>> and Signalroute is that Signalroute allows for the specification of attributes like delivery policy, queuing policy, reliability and delaying policy (Bozga and Olovsky 2002). The UML superstructure specification (OMG 2007b, Clause 9.3.6) and SDL-UML profile specification (ITU-T 2007) show that this is not the case for <<Connector>>.

This issue was not further considered for the transformation hypothesis for the following reason. The AGEDIS IF 2.0 ticket machine example in Appendix F will be used as a reference specification for the verification methods. This IF 2.0 reference specification does not contain instances of Signalroute that own any of the above discussed attributes. Therefore, the comparative analysis on <<Connector>> and Signalroute resulted in the following transformation hypothesis:

Hypothesis 3: a <<Connector>> with an associated InformationFlow that has an association to <<Signal>> can be transformed to a Signalroute for the purpose of automatic test case generation.

4.3.3 Pair – SDL-UML: <<StateMachine>> IF 2.0: Process states

The state machines of the SDL-UML ticket machine model were specified with the following SDL-UML stereotypes: <<StateMachine>>, <<Transition>>, <<Region>>, <<State>> and <<PseudoState>>.

A first comparison of <<StateMachine>> and IF 2.0 state led to the following
result. \textit{StateMachine} is a specialised UML \textit{Behaviour} that can be associated with \textit{activeClass} (OMG 2007b, Figure 13.6), whereby \textit{state} is integrated in the specification of \textit{Process} (Bozga and Olovsky 2002). Both modelling languages have the modelling concept of a “start state”, whereby this is \textit{PseudoState} of kind \textit{initial} in SDL-UML and \textit{state #start} in IF 2.0 (Bozga and Olovsky 2002). In addition, the SDL-UML ticket machine model uses \textit{PseudoState} of kind \textit{choice} that also exists in the IF 2.0 language specification with the same semantics as \textit{state #unstable}. \textit{PseudoState} and \textit{state #unstable} are used in both modelling languages for the transition to more than one target state depending on a condition.

The comparative analysis showed that both modelling languages contain a concept for the transition between states. This concept is implemented in SDL-UML as stereotype \textit{Transition} and in the Intermediated Format as \textit{Transition}. In both modelling languages, a transition can be triggered by the reception of a signal. This is defined in UML with the \textit{ReceiveSignalEvent} (OMG 2007b, Clause 14.3.28) and in IF 2.0 with the modelling concept \textit{input} (Bozga and Olovsky 2002).

The comparative analysis also showed that the reception of a signal can trigger an \textit{Activity} in SDL-UML or an \textit{Action} in IF 2.0. As shown in the state machine model for \textit{activeClass} TicketTray in Figure 21, in \textit{State} \textit{ready} the reception of a \textit{Signal} \textit{EmptyTicketTray} results in the \textit{Activity} \textit{returnFalse}. This triggers the sending of a response signal defined by an action of stereotype \textit{SendSignalAction}. This way of sending an output signal as part of a transition is defined in IF 2.0 with the modelling concept of
output (Bozga and Olvovsky 2002).

The comparative analysis of this section resulted in the following transformation hypothesis:

Hypothesis 4: \(<\text{StateMachine}\>\) can be transformed to instances of \text{state} in IF 2.0 \text{Process}, \(<\text{PseudoState}\>\) of kind \text{initial} can be transformed \text{state } \#\text{start}, \(<\text{State}\>\) can be transformed to \text{state}, \(<\text{PseudoState}\>\) of kind \text{choice} can be transformed to \text{state } \#\text{unstable}, \text{ReceiveSignalEvent} can be transformed to \text{input}, \(<\text{SendSignalAction}\>\) can be transformed to \text{output}.

4.3.4 Pair – SDL-UML: \(<\text{Signal}\>\) IF 2.0: Signal

The comparative analysis showed that \(<\text{Signal}\>\) and IF 2.0 \text{Signal} share similar semantics. As stated in Bozga et al. (2004b), a \text{Signal} can own a number of typed arguments. This is also the case for \(<\text{Signal}\>\) as outlined in the UML superstructure specification (OMG 2007b, Clause 13.3.24). This result of the comparative analysis was formulated in the following transformation hypothesis:

Hypothesis 5: a \(<\text{Signal}\>\) can be transformed to \text{Signal}.

4.3.5 Pair – SDL-UML: \(<\text{PrimitiveType}\>\) IF 2.0: predefined basic data type

According to the SDL-UML profile specification, \(<\text{PrimitiveType}\>\) extends the UML Metaclass \text{PrimitiveType} as specified in OMG (2007b, Clause 7.3.43) “(...) primitive type defines a predefined data type, without any relevant substructure”. Two classes of type \(<\text{PrimitiveType}\>\) \text{PrimitiveType} were
defined for the SDL-UML ticket machine model. The first class had the value 
name = Integer and the second class had the value name = Boolean. The 
comparative analysis showed that the IF 2.0 language contains a number of 
predefined basic data types (Bozga et al. 2004b). These predefined basic data 
types also contain the type Integer. The comparative analysis of this section 
resulted in the following transformation hypothesis:

Hypothesis 6: \(<\text{PrimitiveType}>> PrimitiveType with name = Integer can be 
transformed to Integer.

4.3.6 SDL-UML: \(<\text{Interface}>> and \(<\text{Port}>> with no IF 2.0 equivalent

The comparative analysis revealed that the IF 2.0 language specification did not 
contain the UML modelling concepts represented by \(<\text{Interface}>> and 
\(<\text{Port}>>. These two stereotypes could be used as an alternative to the 
specification of Signalroute as discussed in Section 4.3.2. This alternative 
definition of a Signalroute could be derived from \(<\text{Connector}>> and the 
\(<\text{Signal}>> definitions that are part of \(<\text{Interface}>> defined for each 
\(<\text{Port}>>.

This transformation logic requires the querying of four SDL-UML stereotypes and 
a complex transformation logic. In comparison to this, the discussion in Section 
4.3.2 showed that Signalroute can be derived from just two SDL-UML 
stereotypes, i.e. \(<\text{Connector}>>, \(<\text{Signal}>> and the UML Class 
InformationFlow. This solution is more effective than the one deriving 
Signalroute from \(<\text{Connector}>>, \(<\text{Port}>>, \(<\text{Signal}>> and
Therefore, as a result of the comparative analysis, the SDL-UML stereotypes `<Interface>` and `<Port>` and `<Interfaces>` will not be considered for the transformation of the SDL-UML ticket machine model to the Intermediated Format 2.0.

### 4.4 Specification of MOFScript transformation template

The hypotheses that were formulated as a result of the comparative analysis were implemented as transformation rules in a MOFScript transformation template (Appendix H). The specification of this transformation template generated the following results.

The MOFScript tool could read the XMI based ticket machine model and performed the syntax checking functionality as described in Section 2.5. An example for this syntax checking functionality is given in Figure 24. As shown in this figure, the MOFScript statement `self.activity->forEach(st:uml.State)` referenced an association of type `activity` that was not defined in the abstract syntax of Figure 15.2 in the UML superstructure specification (OMG 2007b). The MOFScript parser detected this incorrect syntax and outputted an error message as shown in Figure 24. This result proved that the MOFScript tool could integrate with the EMF framework as discussed in Section 3.5.
Figure 24  MOFScript tool syntax checking functionality

As shown in Appendix H, transformation rules were specified for each transformation hypothesis.

4.5 MOFScript Transformation

The execution of the transformation template generated the following results. For the first transformation run, the MOFScript transformation template compiled and run without any error messages, but did not produce any output. An analysis of this observation revealed that it was due to two XMI elements in the SDL-UML ticket machine model as shown in Figure 25. These elements were <xmi:Documentation> and <ownedComment> as element of <uml:Model>. A removal of these elements from the XMI ticket machine model enabled to MOFScript tool to generate IF 2.0 system code as shown in Appendix G.
4.6 Verification of transformation hypotheses

This section describes the results of the two verification methods as outlined in Section 3.7.

4.6.1 Results of verification method 1

The first verification method as described in Section 3.7.1 required the entire SDL-UML ticket machine model to be transformed to an IF 2.0 system specification. This transformation required an existing AGEDIS toolset documentation on the various IF 2.0 procedures that the AGEDIS toolset environment supports.

For example, as shown in Appendix F and Figure 26, the AGEDIS IF 2.0 ticket machine example uses the IF 2.0 procedure call

```
if_pid_mk(if_TicketTray_process, 0)
```

for establishing a link between the IF 2.0 processes TicketMachine and TicketTray.
procedure IF_Init_ticketTray;
  fpar out x IF_ConnectivityTable_ticketTray_2;
  {#
    x [0] [0] = if_pid_mk(if_TicketTray_process, 0);
  #}
endprocedure;

Figure 26 IF 2.0 procedure call

It can be concluded that from the context of \texttt{if_pid_mk()}, that his procedure implements the IF 2.0 inter-process communication via process identifiers as outline in Bozga et al. (2004a). An analysis of the AGEDIS toolset documentation revealed that this procedure was not documented. But the transformation of the SDL-UML ticket machine model to an IF 2.0 system specification required a documented AGEDIS IF 2.0 procedure reference.

One could argue that AGEDIS IF 2.0 inter-process communication via explicitly defined signalroutes (Bozga et al. 2004a) would be an alternative to the undocumented AGEDIS IF 2.0 inter-process communication via process identifiers. In order to verify the feasibility of such an approach, an IF 2.0 ticket machine specification as shown in Appendix E was created. This specification was input to the AGEDIS toolset together with test directives. As a result, the AGEDIS toolset stopped the creation of abstract test cases and provided debugging information with no further leading background information.
An error analysis on the debugging information as depicted in Figure 27 revealed the following ambiguity between the AGEDIS toolset documentation (Cavarra and Trost 2002) and the IF 2.0 language specification as outlined in Bozga et al. (2004a), or Bozga and Lakhnech (2002). Cavarra and Trost state in their document that an UML *PseudoState* of type initial is being transformed into an IF 2.0 *#init state*. The IF 2.0 language specifications as described by Bozga et al. (2004a), or Bozga and Lakhnech (2002) only reference the IF 2.0 *#start state*, an IF 2.0 *#init state* is not specified in these documents.

With respect to this ambiguity in the AGEDIS IF 2.0 language implementation, the undocumented IF 2.0 procedure calls provided by the AGEDIS environment and the debugging information without further leading information, this first method could not be used for the verification of the transformation hypotheses.
### 4.6.2 Results of verification method 2

This section describes the results of the second verification method. As shown in Table 5, all SDL-UML stereotypes of the SDL-UML ticket machine model except `<<Connector>>` could be transformed to the Intermediated Format 2.0.

<table>
<thead>
<tr>
<th>Transformation hypothesis</th>
<th>MOFScript transformation rule (Appendix H)</th>
<th>SDL-UML stereotype from SDL-UML ticket machine model</th>
<th>SDL-UML stereotype transformed to IF 2.0 element in MOFScript output (Appendix G)</th>
<th>IF 2.0 element in AGEDIS IF 2.0 ticket machine example (Appendix F)</th>
<th>Covered by abstract test case in Appendix I</th>
<th>Verification result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis 1</td>
<td>Rule 1</td>
<td>outermost <code>&lt;&lt;activeClass&gt;&gt;</code></td>
<td>System TicketMachine; endsystem;</td>
<td>system IF_ModelGeneratedFromAml;_endsystem;</td>
<td>Yes, by specifying the signals that can be sent to and received from the ticket machine system.</td>
<td>Hypothesis verified</td>
</tr>
<tr>
<td>Hypothesis 2</td>
<td>Rule 2</td>
<td><code>&lt;&lt;activeClass&gt;&gt;</code> &gt; <code>&lt;&lt;Property&gt;&gt;</code> &gt; <code>Class of kind Block</code></td>
<td>Process TicketTray(1); endprocess;</td>
<td>process TicketTray (1); endprocess;</td>
<td>Yes, this abstract test case covers all instances of IF 2.0 Process that are specified in the AGEDIS IF 2.0 ticket machine example. This is achieved by specifying the xml elements <code>&lt;member signature=&quot;functionName&quot;&gt;</code> that invoke the respective behaviour in all instances of Process.</td>
<td>Hypothesis verified</td>
</tr>
<tr>
<td>Hypothesis 3</td>
<td>Rule 3</td>
<td><code>&lt;&lt;Connector&gt;&gt;</code></td>
<td>signalroute sig0 from TicketMachine to TicketTray with signal EmptyTicketTray(bool ean); etc.</td>
<td>procedure IF_Init_ticketTray; procedure IF_ticketTray; etc.</td>
<td>No, the abstract test case in Appendix I was generated on basis of inter-process communication via process identifiers, instead of inter-process communication via signalroutes.</td>
<td>Hypothesis not verified</td>
</tr>
<tr>
<td>Hypothesis</td>
<td>Rule</td>
<td>&lt;Signal&gt;</td>
<td>&lt;PrimitiveType&gt;</td>
<td>System</td>
<td>PrimitiveType with name = Integer</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>----------</td>
<td>-----------------</td>
<td>--------</td>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>&lt;&lt;StateMachine&gt;&gt;</td>
<td>&lt;&lt;SendSignalAction&gt;&gt;</td>
<td>ReceiveSignalEvent</td>
<td>Rule, 5, 6, 8, 9: process TicketTray(1); state start #start; nextstate ready; input EmptyTicketTray(booolean); output EmptyTicketTray([false]); nextstate ready; . endprocess;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>&lt;&lt;Signal&gt;&gt;</td>
<td>&lt;&lt;Signal&gt;&gt;</td>
<td>System TicketMachine; signal EmptyTicketTray(booolean); . endsystem;</td>
<td>The signals that are specified in the abstract test case invoke behaviour that requires the value of constants of type integer.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>System COINSTORE_CAP = 6;</td>
<td>Yes, by defining the signals that can be sent to and received from the ticket machine system.</td>
<td></td>
</tr>
</tbody>
</table>

Yes, by sending the signals as specified in the abstract test case and invoking the corresponding behaviours that are specified in the IF 2.0 processes. Hypothesis verified

Table 5 Overview on verification results of transformation hypotheses
4.7 Validation

This section evaluates how the initial research question had been answered from the research that was carried out. The initial research question was formulated as:

*To what extent it is possible to transform SDL-UML stereotypes to modelling concepts of the Intermediated Format 2.0 for the purpose of automatic test case generation?*

This question was answered for the subset of SDL-UML stereotypes that were used for the modelling of the SDL-UML ticket machine as shown in Table 4. The reasons and implications of this result will be discussed in the following analysis section and Chapter 5.

4.8 Analysis

This section will start with a discussion on the results of the research methods. This is followed by an analysis on the effectiveness of the research methods that were performed for this dissertation.

4.8.1 Results of research methods

The research question was only answered for a subset of SDL-UML stereotypes. This was a result of the considerable complexity of the SDL-UML profile specification. A SDL-UML system model that had contained all SDL-UML stereotypes would have exceeded the available time budget for this dissertation. Therefore, only a subset of SDL-UML stereotypes (Table 4) was selected for the specification of the SDL-UML ticket machine model.
As a consequence of this, a number of SDL-UML stereotypes could not be considered for the transformation, although they were also important for the automatic generation of abstract test cases. Examples for these SDL-UML stereotypes are \texttt{<<PassiveClass>>} for the modelling of structured data types, or \texttt{<<Operation>>} for the modelling of communication between objects via remote procedure calls.

For those SDL-UML stereotypes that were used for the specification of the SDL-UML ticket machine model, all stereotypes except \texttt{<<Connector>>} were successfully transformed to the Intermediated Format 2.0. As a result of this, the transformation of the SDL-UML ticket machine model did not allow for the generation of abstract test cases for the following reason. As discussed in Section 1.2, the generation of abstract test cases requires the existence of IF 2.0 signalroutes for the sending of signals to the SUT and for the receiving of signals from the SUT.

4.8.2 Effectiveness of research methods

An analysis on the effectiveness of the research methods showed the following. The first verification method as described in Section 3.7.1 could not be performed as envisaged because of the following problems within the AGEDIS toolset environment.

1. The ambiguity of the AGEDIS IF 2.0 language specification.
2. The limited documentation on IF 2.0 procedure calls within the AGEDIS environment.
3. The no further leading debugging information from the AGEDIS
As result of these problems only the second verification method from Section 3.7.2 could be performed. Therefore, the verification of the transformation hypotheses relied on an existing AGEDIS IF 2.0 ticket machine specification.

The analysis on the effectiveness of the research methods also showed that the comparative analysis depended on the skills of the person performing this analysis. This was due to the fact that the semantics of the SDL-UML profile and the Intermediated Format 2.0 were specified in human language. This introduces a certain level of ambiguity that is also discussed in the work of Harel and Rumpe (2004).

The effectiveness of the specification of the SDL-UML ticket machine model depended on two assumptions. The first assumption was that the SDL-UML ticket machine model would be an equal representation of same ticket machine system that was also modelled by the AGEDIS IF 2.0 ticket machine specification. This assumption could not be verified because the AGEDIS documentation did not contain a specification of the "real-world" ticket machine system. The second assumption was that the SDL-UML ticket machine model was syntactically correct with respect to the SDL-UML profile specification (ITU-T 2007). This assumption could also not be verified because the syntactical rules of the SDL-UML profile were only available in textual form. This could not be read by the validation parser of the MagicDraw modelling tool.
Chapter 5 Conclusions

This chapter reviews the research project in terms of the research objectives and suggest areas for future research.

5.1 Project review

This research project succeeded partly in transforming SDL-UML stereotypes to modelling concepts of the Intermediated Format 2.0. As discussed in the previous analysis section, this was due to the complexity of the SDL-UML profile specification. From this it can be concluded that the initial research question was not focused enough, so that it could be answered within the available time budget. The scope of the transformation from SDL-UML to IF 2.0 should have been narrowed to specific areas of system structure and behaviour like composite structures, or complex state machines.

Nevertheless, the research project succeeded in combining a complex transformation methodology and a model-based test case generation methodology. In addition, the research project succeeded in specifying transformation rules that were successfully executed within the given tool chain. Therefore, the architecture of software components that was specified for this research project can also be used for further research in the area of SDL-UML to IF 2.0 transformations.

The research project showed that the transformation hypotheses could be verified with one of the existing verification methods. The previous analysis section discussed that the verification relied on one AGEDIS IF 2.0 ticket machine example. For future work in this area it would be beneficial to also perform the
first verification method from Section 3.7.1. This would increase the quality of the verification results.

5.2 Future research

Future research is required in order to investigate, whether SDL-UML stereotypes that were not included in Table 4, can be transformed to IF 2.0. This work can be further extended by considering communication between IF 2.0 objects via remote procedure calls or remote variables.

Also the verification of the transformation hypotheses requires future work. A verification of the transformation hypotheses that is based on the transformation of a SDL-UML model to IF 2.0 would considerably facilitate and extend future research. The following work is required in order to perform the first verification method as outlined in Section 3.7.1.

1. The ambiguity of the AGEDIS IF 2.0 language specification needs to be resolved.
2. The documentation on the IF 2.0 procedure calls within the AGEDIS environment needs to be extended.
3. The debugging information from the AGEDIS compiler as shown in Figure 27 needs to be resolved.

Another area for future research is the verification of the hypothesis that the SDL-UML stereotype \(<\text{Connector}\)> can be transformed to the IF 2.0 modelling concept of Signalroute. This would enable the verification of the hypothesis that the complete SDL-UML ticket machine model (Section 4.2) can be transformed to the Intermediated Format 2.0 for the purpose of automatic test case generation.
References


SINTEF (n.d.) *SDL by example* [online], http://www.sintef.no/time/elb40/html/elb/sdl/sdl_t03.htm [Accessed 20 November 2009].


The Open University (2005) M885 Analysis and design of enterprise systems: an object-oriented approach, Revisiting development and further concepts, Milton Keynes, Open University.


Bibliography


Index

A
abstract test case, 4
abstract test case generator, 6
AGEDIS methodology, 12
AGEDIS modelling language, 13
AGEDIS project, 12
AGEDIS software components, 12
AGEDIS-TGV engine, 32

I
IF 2.0 toolset architecture, 19
Intermediated Format 2.0, 7

J
JUni, 5

M
MBT tools, 1
model transformation approaches, 28
Model-based Testing, 1
Model-to-code transformation, 29
MOF Model to Text Transformation Language, 29
MOFScript tool, 30

O
OMEGA profile, 22

S
SDL-UML profile, 11
semantics, 27
Specification and Description Language, 10
syntax, 26

test case, 2
traces, 2

U
UML 2.0 infrastructure specification, 23
UML 2.0 modelling capabilities, 24
UML 2.0 modelling concepts, 23
UML 2.0 superstructure specification, 23
Unified Modelling Language, 10

X
XSL Transformations (XSLT) language, 43
Appendix A: Extended Abstract
Model-based Testing: Transforming SDL-UML models to the Intermediated Format 2.0

Thomas Rogenhofer

Extended abstract of Open University MSc Dissertation submitted 12 February 2010

Introduction

The Unified Modelling Language (UML) 2.0 has become a standard for the modelling of information systems. It provides an extension mechanism that enables the modelling of specific domains like software testing, or real-time systems. This UML 2.0 extension mechanism is also referred to as an UML profile that contains a number of modelling elements that are called stereotypes.

In 2007, the International Telecommunication Union published the SDL-UML profile. This profile is based on the Specification and Description Language (SDL) that represents another standard for the modelling of information systems. Although it is possible to automatically generate test cases from SDL models, this is not the case for SDL-UML models. It was therefore the purpose of this work to investigate whether SDL-UML models could be transformed to an intermediate text based language, the Intermediated Format (IF) 2.0. This transformation allows an existing automatic testing suite, the AGEDIS toolset, to automatically generate test cases from an IF 2.0 system specification.

Results

A first analysis revealed the complexity of the SDL-UML profile in terms of the UML 2.0 modelling concepts that were used for its specification. From this it became obvious that only a subset of SDL-UML stereotypes could be considered for the transformation to IF 2.0. The specification of a SDL-UML ticket machine
model defined this subset of SDL-UML stereotypes. The following figure shows the structure, interfaces and signals that were defined for the SDL-UML ticket machine model.

![Diagram of SDL-UML ticket machine model]

The following table provides a detailed overview on the SDL-UML stereotypes that were used for the specification of the SDL-UML ticket machine model. In addition, this table shows for each SDL-UML stereotype an equivalent IF 2.0 modelling element. For each of these pairs of SDL-UML stereotype and IF 2.0 modelling element a transformation hypothesis was formulated.
<table>
<thead>
<tr>
<th>Part in UML 2.0 superstructure specification</th>
<th>Category - UML 2.0 modelling capability</th>
<th>SDL-UML Stereotype from SDL-UML ticket machine model</th>
<th>IF 2.0 modelling concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part I - Structure</td>
<td>Classes</td>
<td>&lt;&lt;Package&gt;&gt; &lt;&lt;PrimitiveType&gt;&gt;</td>
<td>Not defined&lt;br&gt;Predefined basic data type</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>Composite Structures</td>
<td>&lt;&lt;activeClass&gt;&gt; &lt;&lt;Port&gt;&gt; &lt;&lt;Interface&gt;&gt; &lt;&lt;Connector&gt;&gt; &lt;&lt;Property&gt;&gt;</td>
<td>Process/System&lt;br&gt;Not defined&lt;br&gt;Not defined&lt;br&gt;Signalroute&lt;br&gt;Process</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>Actions</td>
<td>&lt;&lt;sendSignalAction&gt;&gt;</td>
<td>output</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>Activities</td>
<td>&lt;&lt;Activity&gt;&gt; &lt;&lt;SequenceNode&gt;&gt; &lt;&lt;ActivityFinalNode&gt;&gt;</td>
<td>Action&lt;br&gt;Not defined&lt;br&gt;Not defined</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>Common Behaviors</td>
<td>&lt;&lt;Signal&gt;&gt;</td>
<td>Signal</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>State Machines</td>
<td>&lt;&lt;StateMachine&gt;&gt; &lt;&lt;State&gt;&gt; &lt;&lt;PseudoState&gt;&gt; &lt;&lt;Region&gt;&gt; &lt;&lt;Transition&gt;&gt;</td>
<td>Process States&lt;br&gt;State #start, Endstate&lt;br&gt;State #unstable&lt;br&gt;Not defined&lt;br&gt;Transition</td>
</tr>
<tr>
<td>Part III - Supplement</td>
<td>Auxiliary Constructs</td>
<td>Not used in SDL-UML model</td>
<td>Not defined</td>
</tr>
</tbody>
</table>

The verification of the transformation hypotheses showed that all SDL-UML stereotypes except for <<Connector>> could be transformed to their equivalent IF 2.0 modelling concept.

**Analysis**

An analysis of the verification results revealed that the SDL-UML ticket machine model could not be transformed to the Intermediate Format 2.0 for the purpose of automatic test case generation. This was due to the fact that the automatic generation of test cases requires the definition of communication channels for sending and receiving signals from the system under test. These communication channels are specified in SDL-UML with the stereotype <<Connector>> that could not be transformed to IF 2.0.

The analysis on the effectiveness of the research methods showed that the verification methods were constrained by a number of issues related to the AGEDIS toolset. These issues were related to an ambiguity of the AGEDIS IF 2.0 language implementation, or the limited documentation of IF 2.0 procedure calls within the AGEDIS toolset. Without a solution for these issues, future research on the transformation of SDL-UML to IF 2.0 requires considerable more effort.
Discussion

This research project has shown that it is possible to transform a number of SDL-UML stereotypes to the Intermediated Format 2.0. These transformations were supported by an Eclipse-based transformation tool that integrated effectively with the Eclipse Modelling Framework (EMF). Given that the previously discussed issues related to the AGEDIS toolset and the transformation of $\langle\langle\text{Connector}\rangle\rangle$ were resolved, the implemented tool chain would represent an effective architecture for automatic test case generation from SDL-UML models.
Appendix B - Result of analysis of SDL-UML modelling concepts

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Part I - Structure</td>
<td>7.0</td>
<td>Classes</td>
<td>Not applicable</td>
<td>The Classes package contains sub packages that deal with the basic modelling concepts of UML, and in particular classes and their relationships.</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.11</td>
<td>Classes</td>
<td>DataType</td>
<td>A data type is a type whose instances are identified only by their value. A DataType may contain attributes to support the modelling of structured data types.</td>
<td>7.4</td>
<td>Stereotype DataType</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.16</td>
<td>Classes</td>
<td>Enumeration</td>
<td>An enumeration is a data type whose values are enumerated in the model as enumeration literals.</td>
<td>7.5</td>
<td>Stereotype Enumeration</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.18</td>
<td>Classes</td>
<td>Expression</td>
<td>An expression is a structured tree of symbols that denotes a (possibly empty) set of values when evaluated in a context.</td>
<td>10.1</td>
<td>Stereotype Expression</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.23</td>
<td>Classes</td>
<td>InstanceValue</td>
<td>An instance value is a value specification that identifies an instance.</td>
<td>10.2</td>
<td>Stereotype InstanceValue</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.24</td>
<td>Classes</td>
<td>Interface</td>
<td>An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract any instance of a classifier that realizes the interface must fulfil that contract.</td>
<td>7.6</td>
<td>Stereotype Interface</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.26</td>
<td>Classes</td>
<td>LiteralBoolean</td>
<td>A literal Boolean is a specification of a Boolean value.</td>
<td>10.3</td>
<td>Stereotype LiteralBoolean</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.27</td>
<td>Classes</td>
<td>LiteralInteger</td>
<td>A literal integer is a specification of an integer value.</td>
<td>10.4</td>
<td>Stereotype LiteralInteger</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.28</td>
<td>Classes</td>
<td>LiteralNull</td>
<td>A literal null specifies the lack of a value.</td>
<td>10.7</td>
<td>Stereotype LiteralNull</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.30</td>
<td>Classes</td>
<td>LiteralString</td>
<td>A literal string is a specification of a string value.</td>
<td>10.5</td>
<td>Stereotype LiteralString</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.31</td>
<td>Classes</td>
<td>LiteralUnlimitedNatural</td>
<td>A literal unlimited natural is a specification of an unlimited natural number.</td>
<td>10.6</td>
<td>Stereotype LiteralUnlimitedNatural</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.32</td>
<td>Classes</td>
<td>MultiplicityElement</td>
<td>A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound. A multiplicity element embeds this information to specify the allowable cardinalities for an instantiation of this element.</td>
<td>7.12</td>
<td>Stereotype Property</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.36</td>
<td>Classes</td>
<td>Operation</td>
<td>An operation is a behavioural feature of a classifier that specifies the name, type, parameters, and constraints for invoking an associated behavior.</td>
<td>7.7</td>
<td>Stereotype Operation</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.37</td>
<td>Classes</td>
<td>Package</td>
<td>A package is used to group elements, and provides a namespace for the grouped elements.</td>
<td>7.8</td>
<td>Stereotype Package</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.43</td>
<td>Classes</td>
<td>PrimitiveType</td>
<td>A primitive type defines a predefined data type, without any relevant substructure (i.e., it has no parts in the context of UML). A primitive DataType may have algebra and operations defined outside of UML, for example, mathematically.</td>
<td>7.11</td>
<td>Stereotype PrimitiveType</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.44</td>
<td>Classes</td>
<td>Property</td>
<td>A property is a structural feature. A property related to a classifier by ownedAttribute represents an attribute, and it may also represent an association end. A property related to an Association by memberEnd or its specializations represents an end of the association.</td>
<td>7.12</td>
<td>Stereotype Property</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.49</td>
<td>Classes</td>
<td>StructuralFeature</td>
<td>A structural feature is a typed feature of a classifier that specifies the structure of instances of the classifier.</td>
<td>7.12</td>
<td>Stereotype Property</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.5</td>
<td>Classes</td>
<td>BehavioralFeature</td>
<td>A behavioural feature is a feature of a classifier that specifies an aspect of the behavior of its instances.</td>
<td>7.7</td>
<td>Stereotype Operation</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.52</td>
<td>Classes</td>
<td>TypedElement</td>
<td>A typed element is an element that has a type that serves as a constraint on the range of values the element can represent. Typed element is an abstract metaclass.</td>
<td>7.12</td>
<td>Stereotype Property</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.54</td>
<td>Classes</td>
<td>ValueSpecification</td>
<td>A value specification is the specification of a (possibly empty) set of instances, including both objects and data values.</td>
<td>10.8</td>
<td>Stereotype ValueSpecification</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.6</td>
<td>Classes</td>
<td>BehavioredClassifier</td>
<td>A BehavioredClassifier may have an interface realization.</td>
<td>7.1</td>
<td>Stereotype ActiveClass</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>7.3.7</td>
<td>Classes</td>
<td>Class</td>
<td>A class describes a set of objects that share the same specifications of features, constraints, and semantics.</td>
<td>7.1</td>
<td>Stereotype ActiveClass</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>9.0</td>
<td>Composite Structures</td>
<td>Not applicable</td>
<td>The term “structure” in this clause refers to a composition of interconnected elements, representing run-time instances collaborating over communications links to achieve some common objectives.</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
| Part I - Structure | 9.3.1 | Composite Structures | Class | Extends the metaclass Class with the capability to have an internal structure and ports. | 7.1 | Stereotype ActiveClass  
Stereotype Class  
Stereotype PassiveClass |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Part I - Structure</td>
<td>9.3.11</td>
<td>Composite Structures</td>
<td>Port</td>
<td>A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts.</td>
<td>7.10</td>
<td>Stereotype Port</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>9.3.6</td>
<td>Composite Structures</td>
<td>Connector</td>
<td>Specifies a link that enables communication between two or more instances.</td>
<td>7.3</td>
<td>Stereotype Connector</td>
</tr>
<tr>
<td>Part I - Structure</td>
<td>9.3.7</td>
<td>Composite Structures</td>
<td>ConnectorEnd</td>
<td>A connector end is an endpoint of a connector, which attaches the connector to a connectable element. Each connector end is part of one connector.</td>
<td>7.3</td>
<td>Stereotype Connector</td>
</tr>
</tbody>
</table>
| Part I - Structure | 9.3.8  | Composite Structures | EncapsulatedClassifier | Extends a classifier with the ability to own ports as specific and type checked interaction points. | 7.1 | Stereotype ActiveClass  
Stereotype Class  
Stereotype PassiveClass |
### Part II - Behavior

#### 11.3.10 Actions CallOperationAction

CallOperationAction is an action that transmits an operation call request to the target object, where it may cause the invocation of associated behavior. The argument values of the action are available to the execution of the invoked behavior.

#### 9.6 Stereotype CallOperationAction

<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.16 Actions CreateObjectAction</th>
</tr>
</thead>
</table>

CreateObjectAction is an action that creates an object that conforms to a statically specified classifier and puts it on an output pin at runtime.

#### 9.9 Stereotype CreateObjectAction

<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.26 Actions Break</th>
</tr>
</thead>
</table>

The stereotype Break is a concrete subtype of the stereotype OpaqueAction.

#### 9.6 Stereotype Break

<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.26 Actions Continue</th>
</tr>
</thead>
</table>

The stereotype Continue is a concrete subtype of the stereotype OpaqueAction.

#### 9.8 Stereotype Continue

<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.26 Actions Empty</th>
</tr>
</thead>
</table>

The stereotype Empty is a concrete subtype of the stereotype OpaqueAction.

#### 9.10 Stereotype Empty

<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.26 Actions ExpressionAction</th>
</tr>
</thead>
</table>

The stereotype ExpressionAction is a concrete subtype of the stereotype OpaqueAction.

#### 9.12 Stereotype ExpressionAction

<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.26 Actions OpaqueAction</th>
</tr>
</thead>
</table>

An action with implementation-specific semantics.

#### 9.16 Stereotype OpaqueAction
<p>| Part II - Behavior | 11.3.26 | Actions | ResetAction | A timer is cancelled with a reset action represented by a ResetAction stereotype. The ResetAction stereotype is a concrete subtype of the stereotype OpaqueAction. | 9.17 | Stereotype ResetAction |
| Part II - Behavior | 11.3.26 | Actions | SetAction | A timer is started with a set action represented by a SetAction stereotype. The SetAction stereotype is a concrete subtype of the stereotype OpaqueAction. | 9.21 | Stereotype SetAction |
| Part II - Behavior | 11.3.45 | Actions | SendSignalAction | SendSignalAction is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity. | 9.20 | Stereotype SendSignalAction |
| Part II - Behavior | 11.3.47 | Actions | StructuralFeatureAction | This abstract action class statically specifies the structural feature being accessed. | 9.3 | Stereotype AddStructuralFeatureValueAction Stereotype CreateObjectAction |
| Part II - Behavior | 11.3.5 | Actions | AddStructuralFeatureValueAction | AddStructuralFeatureValueAction is a write structural feature action for adding values to a structural feature. | 9.3 | Stereotype AddStructuralFeatureValueAction Stereotype CreateObjectAction |</p>
<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>11.3.52</th>
<th>Actions</th>
<th>VariableAction</th>
<th>VariableAction is an abstract class for actions that operate on a statically specified variable.</th>
<th>9.4</th>
<th>AddVariableValueAction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part II - Behavior</td>
<td>11.3.6</td>
<td>Actions</td>
<td>AddVariableValueAction</td>
<td>AddVariableValueAction is a write variable action for adding values to a variable.</td>
<td>9.4</td>
<td>9.9</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.17</td>
<td>Activities</td>
<td>Clause</td>
<td>A clause is an element that represents a single branch of a conditional construct, including a test and a body section. The body section is executed only if (but not necessarily if) the test section evaluates true.</td>
<td>9.7</td>
<td>ConditionalNode</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.35</td>
<td>Activities</td>
<td>For</td>
<td>The stereotype For is a concrete subtype of the stereotype LoopNode.</td>
<td>9.13</td>
<td>For</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.35</td>
<td>Activities</td>
<td>LoopNode</td>
<td>A loop node is a structured activity node that represents a loop with setup, test, and body sections.</td>
<td>9.15</td>
<td>LoopNode</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.35</td>
<td>Activities</td>
<td>While</td>
<td>The stereotype While is a concrete subtype of the stereotype LoopNode.</td>
<td>9.23</td>
<td>While</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.4</td>
<td>Activities</td>
<td>Activity</td>
<td>An activity is the specification of parameterized behavior as the coordinated sequencing of subordinate units whose individual elements are actions. There are actions that invoke activities (directly by &quot;CallBehaviorAction (from BasicActions)&quot; on page 244 or indirectly as methods by &quot;CallOperationAction (from BasicActions)&quot; on page 246).</td>
<td>9.1 Stereotype Activity</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------</td>
<td>------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.47</td>
<td>Activities</td>
<td>SequenceNode</td>
<td>A sequence node is a structured activity node that executes its actions in order.</td>
<td>9.19 Stereotype SequenceNode</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.6</td>
<td>Activities</td>
<td>Return</td>
<td>The stereotype Return is a concrete subtype of the stereotype ActivityFinalNode.</td>
<td>9.18 Stereotype Return</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.6</td>
<td>Activities</td>
<td>Stop</td>
<td>The stereotype Stop is a concrete subtype of the stereotype ActivityFinalNode.</td>
<td>9.22 Stereotype Stop</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.6</td>
<td>Activities</td>
<td>ActivityFinalNode</td>
<td>An activity final node is a final node that stops all flows in an activity.</td>
<td>9.2 Stereotype ActivityFinalNode</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.18</td>
<td>Activities</td>
<td>ConditionalNode</td>
<td>A conditional node is a structured activity node that represents an exclusive choice among some number of alternatives</td>
<td>9.7 Stereotype ConditionalNode</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.18</td>
<td>Activities</td>
<td>Decision</td>
<td>The stereotype Decision is a concrete subtype of the stereotype ConditionalNode.</td>
<td>9.11 Stereotype Decision</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>12.3.18</td>
<td>Activities</td>
<td>13.4.18</td>
<td>Activities</td>
<td>13.4.18</td>
<td>Activities</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>------------</td>
<td>---------</td>
<td>------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If</td>
<td></td>
<td>If</td>
<td></td>
<td>If</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The stereotype If is a concrete subtype of the stereotype ConditionalNode.</td>
<td></td>
<td>The Common Behaviors packages specify the core concepts required for dynamic elements and provide the infrastructure to support more detailed definitions of behavior.</td>
<td></td>
<td>Not applicable</td>
</tr>
<tr>
<td>13</td>
<td>Common Behaviours</td>
<td>Not applicable</td>
<td>Common Behaviours</td>
<td>Interface</td>
<td>Interface</td>
<td>Interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Common Behaviors packages specify the core concepts required for dynamic elements and provide the infrastructure to support more detailed definitions of behavior.</td>
<td></td>
<td>Adds the capability for interfaces to include receptions (in addition to operations).</td>
<td></td>
<td>Adds the capability for interfaces to include receptions (in addition to operations).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Behavior is a specification of how its context classifier changes state over time.</td>
<td></td>
<td>An operation may invoke both the execution of method behaviors as well as other behavioural responses.</td>
<td></td>
<td>An operation may invoke both the execution of method behaviors as well as other behavioural responses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A signal is a specification of send request instances communicated between objects.</td>
<td></td>
<td>A signal is a specification of send request instances communicated between objects.</td>
<td></td>
<td>A signal is a specification of send request instances communicated between objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stereotype If</td>
<td></td>
<td>Stereotype If</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interface</td>
<td></td>
<td>Interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Behavior</td>
<td></td>
<td>Behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Operation</td>
<td></td>
<td>Operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Signal</td>
<td></td>
<td>Signal</td>
</tr>
</tbody>
</table>

99
<p>| Part II - Behavior | 13.3.25 | Common Behaviours | Signal Event | A signal event represents the receipt of an asynchronous signal instance. A signal event may, for example, cause a state machine to trigger a transition. | 8.6 | Stereotype Transition |
| Part II - Behavior | 13.3.3 | Common Behaviours | BehavioralFeature | A behavioural feature is implemented (realized) by a behavior. A behavioural feature specifies that a classifier will respond to a designated request by invoking its implementing method. | 7.7 | 8.6 | Stereotype Operation Stereotype Transition |
| Part II - Behavior | 13.3.31 | Common Behaviours | Trigger | A trigger relates an event to a behavior that may affect an instance of the classifier. | 8.6 | Stereotype Transition |
| Part II - Behavior | 13.3.4 | Common Behaviours | BehavioredClassifier | A classifier can have behavior specifications defined in its namespace. One of these may specify the behavior of the classifier itself. | 7.1 | 7.2 | 7.9 | 8.5 | Stereotype ActiveClass Stereotype Class Stereotype PassiveClass Stereotype StateMachine |
| Part II - Behavior | 13.3.8 | Common Behaviours | Class | A class may be designated as active (i.e., each of its instances having its own thread of control) or passive (i.e., each of its instances executing within the context of some other object). | 7.1 | 7.2 | 7.9 | Stereotype ActiveClass Stereotype Class Stereotype PassiveClass |</p>
<table>
<thead>
<tr>
<th>Part II - Behavior</th>
<th>15.3.1</th>
<th>State Machines</th>
<th>ConnectionPointReference</th>
<th>A connection point reference represents a usage (as part of a submachine state) of an entry/exit point defined in the StateMachine reference by the submachine state.</th>
<th>8.6</th>
<th>Stereotype Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part II - Behavior</td>
<td>15.3.10</td>
<td>State Machines</td>
<td>Region</td>
<td>A region is an orthogonal part of either a composite state or a state machine. It contains states and transitions.</td>
<td>8.3</td>
<td>Stereotype Region</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.11</td>
<td>State Machines</td>
<td>State</td>
<td>A state models a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur.</td>
<td>8.4</td>
<td>Stereotype State</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.12</td>
<td>State Machines</td>
<td>StateMachine</td>
<td>State machines can be used to express the behavior of part of a system. Behavior is modelled as a traversal of a graph of state nodes interconnected by one or more joined transition arcs that are triggered by the dispatching of series of (event) occurrences. During this traversal, the state machine executes a series of activities associated with various elements of the state machine.</td>
<td>8.5</td>
<td>Stereotype StateMachine</td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.14</td>
<td>State Machines</td>
<td>Transition</td>
<td>A transition is a directed relationship between a source vertex and a target vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to an occurrence of an event of a particular type.</td>
<td>8.6 Stereotype Transition</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>----------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.16</td>
<td>State Machines</td>
<td>Vertex</td>
<td>A vertex is an abstraction of a node in a state machine graph. In general, it can be the source or destination of any number of transitions.</td>
<td>8.4 Stereotype Region</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.2</td>
<td>State Machines</td>
<td>FinalState</td>
<td>A special kind of state signifying that the enclosing region is completed. If the enclosing region is directly contained in a state machine and all other regions in the state machine also are completed, then it means that the entire state machine is completed.</td>
<td>8.1 Stereotype FinalState</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.8</td>
<td>State Machines</td>
<td>PseudoState</td>
<td>A PseudoState is an abstraction that encompasses different types of transient vertices in the state machine graph.</td>
<td>8.2 Stereotype PseudoState</td>
<td></td>
</tr>
<tr>
<td>Part II - Behavior</td>
<td>15.3.9</td>
<td>State Machines</td>
<td>PseudoStateKind</td>
<td>PseudoStateKind is an enumeration type.</td>
<td>8.2 Stereotype PseudoState</td>
<td></td>
</tr>
<tr>
<td>Part III - Supplement</td>
<td>17</td>
<td>Auxiliary Constructs</td>
<td>Not applicable</td>
<td>This clause defines mechanisms for information flows, models, primitive types, and templates.</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----</td>
<td>----------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Part III - Supplement</td>
<td>17.2</td>
<td>Auxiliary Constructs</td>
<td>InformationFlow</td>
<td>The InformationFlow package provides mechanisms for specifying the exchange of information between entities of a system at a high level of abstraction.</td>
<td>7.3</td>
<td>Stereotype Connector</td>
</tr>
</tbody>
</table>
Appendix C - Results of comparative analysis

<table>
<thead>
<tr>
<th>Name of stereotype (ITU-T 2007)</th>
<th>SDL-UML stereotype identifier</th>
<th>SDL-UML stereotype element</th>
<th>SDL-UML stereotype element description</th>
<th>Comment</th>
<th>IF 2.0 modelling concept description</th>
<th>IF 2.0 Element BNF Specification</th>
<th>Hypothesis on transformability for automatic test case generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>Tet</td>
<td>Stereotype</td>
<td>The stereotype ActiveClass is a concrete subtype of the stereotype Class.</td>
<td>Please see each following row in this column for 7.1.</td>
<td>System or Process</td>
<td>NA</td>
<td>Hypothesis 1: an outermost &lt;&lt;activeClass&gt;&gt; can be transformed to a System for the purpose of automatic test case generation. Hypothesis 2: an &lt;&lt;activeClass&gt;&gt; &lt;&lt;Property&gt;&gt; Class of kind Block within the enclosing &lt;&lt;activeClass&gt;&gt; TicketMachine can be transformed to a Process for the purpose of automatic test case generation.</td>
</tr>
</tbody>
</table>
### Attribute

**7.1-a1** Attribute `isConcurrent`:

| `isConcurrent`: Boolean determines the concurrency semantics of an active class. If `isConcurrent` is `false`, all contained instances execute interleaved. If `isConcurrent` is `true`, contained instances execute concurrently, provided they are not also contained in an instance for which `isConcurrent` is `false`. In the IF 2.0 execution model, Process instances execute asynchronously with their own input buffer and local variables. The concept of interleaved execution is not defined in IF 2.0. IF 2.0 Processes do not contain other IF 2.0 Processes. |
|---|---|---|
| NA | NA | NA |

### Constraint

**7.1-c2** Constraint (*If isConcurrent is false, isConcurrent of any contained instance shall be false.*):

| | This constraint cannot be transformed because IF 2.0 Processes execute concurrently. |
| | NA |
| | NA | NA |

**7.1-c3** Constraint (*If the `<ActiveClass>` Class has a `classifierBehavior`, it shall be a `StateMachine`. Behavior of IF 2.0 Processes is specified with the state machine of each Process*):

<p>| | | Process state machine |
| | | NA | NA |</p>
<table>
<thead>
<tr>
<th>Class</th>
<th>Constraint</th>
<th>Type of agent</th>
<th>Principle of replication</th>
<th>Process state machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-c4</td>
<td>If an &lt;&lt;ActiveClass&gt;&gt; Class has a classBehavior and it has a super Class that is another &lt;&lt;ActiveClass&gt;&gt; Class that also has a classBehavior, theStateMachine of the sub-class shall redefine the StateMachine of the super Class. The reason is that in SDL the state machines of agents automatically extend each other, whereas this is not the case in UML.</td>
<td>The principle of replication as discussed in Graf et al. (2006b) shall also be applied to the redefinition of state machines through inheritance.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-c5</td>
<td>An &lt;&lt;ActiveClass&gt;&gt; Class used as the type of a composite property object (of another &lt;&lt;ActiveClass&gt;&gt; Class) shall have isAbstract == false (that is a type based agent in an agent type shall not be based on an abstract type).</td>
<td>IF 2.0 Processes do not contain other IF 2.0 processes, hence this constraint cannot be transformed.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-c6</td>
<td>Constraint</td>
<td>An ownedAttribute that has a type that is a &lt;&lt;ActiveClass&gt;&gt; Class and where aggregationKind == composite shall not have public visibility (an agent instance set cannot be made visible outside the enclosing agent type).</td>
<td>IF 2.0 Processes do not contain other IF 2.0 processes, hence this constraint cannot be transformed.</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-c7</td>
<td>Constraint</td>
<td>A nestedClassifier shall not have public visibility (an agent type, data type, interface type or signal definition cannot be made visible outside the enclosing agent type).</td>
<td>Ad agent-type: IF 2.0 process cannot contain other processes. Ad data types: in IF 2.0, communication between processes can be achieved via shared variables that are visible outside the process. Ad signal definition: in IF 2.0, signal definitions are made on system level. Ad interface: the concept of interfaces does not exist in IF 2.0.</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-c8</td>
<td>Constraint</td>
<td>An ownedConnector shall not have public visibility (a channel cannot be made visible outside the enclosing agent type that owns the channel).</td>
<td>Signalroutes are owned in IF 2.0 by the system and therefore visible for all processes contained within a system specification.</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-c11</td>
<td>Constraint</td>
<td>An ownedBehavior shall only contain a StateMachine.</td>
<td>This complies with the specification of state machines in IF 2.0 Processes.</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-a1</td>
<td>Semantics</td>
<td>An &lt;&lt;ActiveClass&gt;&gt; Class is mapped to an Agent-type-definition.</td>
<td>The &lt;&lt;ActiveClass&gt;&gt; can be Either IF 2.0 Process or System.</td>
</tr>
</tbody>
</table>
either mapped to an IF 2.0 system or a process definition. Clarify how to handle the missing system definition. Because of their semantic execution model, Process Agents cannot be mapped to IF 2.0 Processes.

<p>| system-component | system-component ::= process-decl | signal-route-decl | signal-decl | procedure-decl | var-decl | type-decl | const-decl |
| process-decl ::= process-id ( const ) [ fpar fpar-decl { , fpar-decl } * ] ( process-component ) * endprocess |
| process-component ::= state | procedure-decl | var-decl | type-decl | const-decl |</p>
<table>
<thead>
<tr>
<th>Class</th>
<th>Semantics</th>
<th>Agent type</th>
<th>Either IF 2.0</th>
<th>NA</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-s2</td>
<td>Semantics</td>
<td>System is not defined on SDL-UML profile.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-s3</td>
<td>Semantics</td>
<td>Due to the execution semantics of IF 2.0 only SDL-UML agent types with isConcurrent==true can be mapped, or agent types with isConcurrent==false, if they do not contain other processes.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-s4</td>
<td>Semantics</td>
<td>Under discussion</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Section</td>
<td>Semantics</td>
<td>Example</td>
<td>IF 2.0</td>
<td>Process</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>7.1-s5</td>
<td>Semantics</td>
<td>A nestedClassifier that is an &lt;&lt;ActiveClass&gt;&gt; Class maps to an element of the Agent-type definition-set of the Agent-type-definition.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>7.1-s6</td>
<td>Semantics</td>
<td>A nestedClassifier that is a &lt;&lt;PassiveClass&gt;&gt; Class maps to an Object data-type-definition that is an element of the Data-type-definition-set of the Agent-type-definition.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>7.1-s7</td>
<td>Semantics</td>
<td>A nestedClassifier that is DataType maps to a Value-data-type-definition that is an element of the Data-type-definition-set of the Agent-type-definition.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>7.1-s8</td>
<td>Semantics</td>
<td>A nestedClassifier that is an Interface maps to an Interface-type-definition that is an element of the Data-type-definition-set of the Agent-type-definition.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-s9</td>
<td>Semantics</td>
<td>A nestedClassifier that is a Signal maps to a Signal-definition that is an element of the Signal-definition-set of the Agent-type-definition.</td>
<td>The concept of signals also exists in IF 2.0.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;ActiveClass&gt;&gt;</td>
<td>7.1-s10</td>
<td>Semantics</td>
<td>An ownedAttribute that maps to a Variable-definition (see clause 7.12, Property) is an element of the Variable-definition-set of the Agent-type-definition. An ownedAttribute that is visible outside the &lt;&lt;ActiveClass&gt;&gt; Class (public visibility) and that has a type that is a DataType or &lt;&lt;PassiveClass&gt;&gt; Class is the Variable-definition for an exported variable and also maps to an implicit Signal-definition pair for accessing this exported variable in the defining context of the Agent-type-definition.</td>
<td>In IF 2.0, processes can own variables and processes can communicate via remote variables.</td>
<td>NA</td>
</tr>
<tr>
<td>Page</td>
<td>Semantics</td>
<td>Notes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.1-s11</td>
<td>An ownedAttribute that maps to an Agent-definition (see clause 7.12, Property) is an element of the Agent-definition-set of the Agent-type-definition.</td>
<td>Due to the execution semantics of IF 2.0 only SDL-UML agent types with isConcurrent==true can be mapped, or agent types with isConcurrent==false at system level, if they do not contain other processes.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.1-s12</td>
<td>Each Connector of the ownedConnector maps to an element of the Channel-definition-set of the Agent-type-definition.</td>
<td>NA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.1-s13</td>
<td>Each Port of the ownedPort maps to an element of the Gate-definition-set of the Agent-type-definition.</td>
<td>The concept of ports does not exist in IF 2.0. It should be discussed whether the UML concept of ports should be flattened across the CETA model as it is done with the UML concept of inheritance.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Subsection</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 7.1-s14 | Semantics | Each Behavior of the ownedBehavior maps to an element of either the Composite-state-type-definition-set or the Procedure-definition-set. If the owned Behavior is the method of an Operation, it is an element of the Procedure-definition-set; otherwise it is an element of the Composite-state-type-definition-set. The StateMachine that is the Behavior of the optional classifierBehavior maps to the State-machine-definition of the Agent-type-definition (see clause 8.5, StateMachine). The name of the optional classifierBehavior is mapped to the State-name of the State-machine-definition. The Composite-state-type-identifier of this State-machine-definition identifies the Composite-state-type derived from the StateMachine that is the classifierBehavior.

<p>|  |  | Composite states do not exist in IF 2.0. The SDL-UML ticket machine model does not contain composite states. |</p>
<table>
<thead>
<tr>
<th>&lt;&lt;ActiveClass&gt;&gt;</th>
<th>7.1-s15</th>
<th>Semantics</th>
<th>The ownedParameter set of the Behavior of classifierBehavior maps to the Agent-formal-parameter list of the Agent-type-definition.</th>
<th>An ownedParameter set does not exist in IF 2.0. The SDL-UML ticket machine model does not contain composite states.</th>
<th>NA</th>
<th>NA</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Connector&gt;&gt;</td>
<td>7.3</td>
<td>Metaclass</td>
<td>The stereotype Connector extends the metaclass Connector with multiplicity [1..1].</td>
<td>This maps to the IF 2.0 concept of signalroutes.</td>
<td>Signalroute</td>
<td>signalroute-decl := signalroute signalroute-id { const } ( signalroute-option</td>
<td>* from { process-id</td>
</tr>
<tr>
<td>&lt;&lt;Connector&gt;&gt;</td>
<td>7.3-a1</td>
<td>Attribute</td>
<td>delay: Boolean</td>
<td>This concept also exists for the definition of signalroutes.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Connector&gt;&gt;</td>
<td>7.3-c1</td>
<td>Constraint</td>
<td>In the case of an InformationItem associated with an InformationFlow associated with a Connector, the represented property</td>
<td>This concept also exists for the definition of signalroutes, whereby it is only possible</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
of the InformationItem shall be a Signal or an Operation or an Interface. to associate a Signal with a signalroute.

| <<Connector>> | 7.3-c2 Constraint | There shall always be exactly 2 end properties. | In IF 2.0 the end properties are defined by the processes at the end of each signalroute. | NA | NA | NA |
| <<Connector>> | 7.3-c3 Constraint | A ConnectorEnd that is part of the end property shall have empty lowerValue and upperValue properties. | In IF 2.0 the end properties are defined by the processes at the end of each signalroute. | NA | NA | NA |
| <<Connector>> | 7.3-c4 Constraint | The role property of a ConnectorEnd that is part of the end property of the Connector shall be a Port. | In IF 2.0 the end properties are defined by the processes at the end of each signalroute. | NA | NA | NA |
| <<Connector>> | 7.3-c5 Constraint | The Type property shall be empty. | This concept does not exist in IF 2.0. | NA | NA | NA |
| <<Connector>> | 7.3-c6 Constraint | The redefinedConnector property shall be empty. | This concept does not exist in IF 2.0. | NA | NA | NA |
| <<Connector>> | 7.3-c7 Constraint | The isStatic property shall be false. | This concept does not exist in IF 2.0. | NA | NA | NA |
| <<Connector>> | 7.3-c8 Constraint | There shall be at least one InformationFlow associated with a Connector. | Signalroutes can also be defined without a signal | NA | NA | NA |
| <<Connector>> | 7.3-s1 | Semantics | A <<Connector>>
Connector maps to a Channel-definition. | This maps to the IF 2.0 concept of signalroutes. | NA | signalroute-decl ::= signalroute signalroute-id | const | () signalroute-option | * from { process-id | env } to { process-id | env } with signal-id | , signal-id | * | NA |

<p>| &lt;&lt;Connector&gt;&gt; | 7.3-s2 | Semantics | The name attribute defines the Channel-name. | This maps to the signalroute-id. | NA | signalroute-decl ::= signalroute signalroute-id | const | () signalroute-option | * from { process-id | env } to { process-id | env } with signal-id | , signal-id | * | NA |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Semantics</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3-s3</td>
<td>Semantics</td>
<td>An InformationFlow associated with a Connector defines the Signal-identifier-set of a Channel-path as follows. The conveyed InformationItem set of each InformationFlow defines the Signal-identifier-set of the Channel-path. If the InformationItem set is omitted, then the Signal-identifier-set is computed based on the realized and required interface of the attached Port. If the InformationFlow conveys an Interface, then the Signal-identifier-set is computed according to the transformation rules of Z.100 (see clause 7.6, Interface).</td>
<td>This semantics does not exist in IF 2.0.</td>
</tr>
<tr>
<td>7.3-s4</td>
<td>Semantics</td>
<td>InformationFlow in one direction only (with or without InformationItem) implies that the channel is unidirectional.</td>
<td>In IF 2.0 signalroutes are always unidirectional.</td>
</tr>
<tr>
<td>7.3-s5</td>
<td>Semantics</td>
<td>InformationFlow in both directions (with or without InformationItem) implies that the channel is bidirectional.</td>
<td>In IF 2.0 signalroutes are always unidirectional.</td>
</tr>
<tr>
<td>&lt;&lt;Connector&gt;&gt;</td>
<td>7.3-s6</td>
<td>Semantics</td>
<td>This semantics does not exist in IF 2.0.</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>&lt;&lt;Connector&gt;&gt;</td>
<td>7.3-s7</td>
<td>Semantics</td>
<td>This semantics does not exist in IF 2.0.</td>
</tr>
<tr>
<td>&lt;&lt;Connector&gt;&gt;</td>
<td>7.3-s8</td>
<td>Semantics</td>
<td>This semantics does not exist in IF 2.0.</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>7.11</td>
<td>Stereotype</td>
<td>PrimitiveType</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>7.11-c1</td>
<td>Constraint</td>
<td>The ownedAttribute set shall be empty.</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>7.11-c2</td>
<td>Constraint</td>
<td>The name shall be one of the following: Boolean, Integer, UnlimitedNatural, Character, Charstring, Real, Duration, Time, Bit, Bitstring, Octet, Octetstring or Pid.</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>7.11-c3</td>
<td>Constraint</td>
<td>The generalization property shall be empty.</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>7.11-c4</td>
<td>Constraint</td>
<td>Each ownedOperation association shall specify one of the operations defined for the specific data type (see clause 12, Predefined Data, clauses 12.1.6 and D.3 of [ITU-T Z.100]).</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td>7.11-s1</td>
<td>Semantics</td>
<td>Predefined basic data types do not own operations in IF 2.0.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>-----------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;&lt;PrimitiveType&gt;&gt;</td>
<td></td>
<td>If no ownedOperation associations are defined, each &lt;&lt;PrimitiveType&gt;&gt; PrimitiveType is mapped to a predefined Syntype-definition or a predefined Value-data-type-definition as detailed in the next paragraph. All the contents of the Value-data-type-definition (such as the Literal-signature-set) are implied from the mapping to the specific definition of the SDL &lt;&lt;package Predefined&gt;&gt; item as further defined in clause 12.1. The corresponding items (such as ownedBehavior for the operations) are implied in the SDL-UML meta-model and therefore can be used in expressions.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>7.12 Stereotype Property</td>
<td>The concept of properties in terms of the UML 2.0 composite structures does not exist in IF 2.0. Nevertheless, properties in terms of UML 2.0 classes can be flattened out to IF 2.0 processes.</td>
<td>NA</td>
</tr>
</tbody>
</table>

| <<Property>> | 7.12-a1 Attribute InitialNumber: UnlimitedNatural [0..1] defines the initial number of instances created when an instance of the containing classifier is created. | Also in IF 2.0 the concept of the initial number of instances of a process exists. | NA | NA |

| <<Property>> | 7.12-a2 Attribute ReferenceSort: Boolean determines the treatment of a variable or field as a value sort or reference sort and has a default value false. | This attribute does not exist in IF 2.0. | NA | NA |

<p>| &lt;&lt;Property&gt;&gt; | 7.12-c1 Constraint The aggregation shall not be shared. | This constraint does not exist in IF 2.0. | NA | NA |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>7.12-c2</th>
<th>Constraint</th>
<th>An IF 2.0 process does not have a type.</th>
<th>NA</th>
<th>NA</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>7.12-c3</td>
<td>Constraint</td>
<td>The type shall not be omitted.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Property</td>
<td>7.12-c4</td>
<td>Constraint</td>
<td>If the upperValue is omitted, the lowerValue shall also be omitted.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Property</td>
<td>7.12-c5</td>
<td>Constraint</td>
<td>If the lowerValue is included, the lowerValue shall also be included.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Property</td>
<td>7.12-c6</td>
<td>Constraint</td>
<td>If the type is an ActiveClass, the lowerValue shall also be included.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Property</td>
<td>7.12-c7</td>
<td>Constraint</td>
<td>If the upperValue value is greater than 1 and isOrdered is true, isUnique shall be false, because there is not a predefined SDL data type that is ordered and requires each of its elements to have unique values.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Property</td>
<td>7.12-c8</td>
<td>Constraint</td>
<td>The initialNumber shall be included only if the type is an</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
The value of the InitialNumber shall not be greater than the upperValue.

<table>
<thead>
<tr>
<th>&lt;&lt;Property&gt;&gt;</th>
<th>7.12-c9</th>
<th>Constraint</th>
<th>NA</th>
<th>NA</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>7.12-c10</td>
<td>Constraint</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>7.12-c11</td>
<td>Constraint</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>7.12-c12</td>
<td>Constraint</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>7.12-s1</td>
<td>Semantics</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
Variable-name. The defaultValue defines the Constant-expression. The Sort-reference-identifier is the Sort-identifier of the sort derived from the type property. The Sort-identifier is determined as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>Semantics</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.12-s2</td>
<td>- If there is no upperValue and no lowerValue, the name of the type maps to the SortIdentifier. This concept does not exist in the IF 2.0 language specification.</td>
<td>NA</td>
</tr>
<tr>
<td>7.12-s3</td>
<td>- Otherwise, the Sort-identifier identifies an anonymous sort formed from the SDL predefined Bag (if isOrdered is false and isUnique is false) or Powerset (if isOrdered is false and isUnique is true) or String (if isOrdered is true) DataType instantiated with the sort given by the type as the ItemSort. The anonymous sort is a Value-data-type-definition or Syntype-definition in the same context as the Variable-definition. If the upperValue value is omitted or the lowerValue value.</td>
<td>NA</td>
</tr>
</tbody>
</table>
is zero and the upperValue value is unlimited (* in the concrete syntax), there are no size constraints and the anonymous sort is a Value-data-type-definition with its components derived from the instantiated predefined data type. Otherwise the lowerValue value and upperValue value map to a Range-condition of the anonymous sort, which is a Syntype-definition. The Parent-sort-identifier of this Syntype definition is a reference to another anonymous sort that is the Value-data-type-definition derived in the same way as the case with no size constraints.

<table>
<thead>
<tr>
<th>&lt;&lt;Property&gt;&gt;</th>
<th>7.12-s4</th>
<th>Semantics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If isReadOnly is true, the type is required to be either a DataType (which includes PrimitiveType and Enumeration) or a &lt;&lt;PassiveClass&gt;&gt; Class. When isReadOnly is true, the &lt;&lt;Property&gt;&gt; Property is mapped to a</td>
<td>This concept does not exist in the IF 2.0 language specification.</td>
</tr>
</tbody>
</table>
Constant-expression each time the Property is used in an expression. The defaultValue defines the Constant-expression.

<table>
<thead>
<tr>
<th>Property</th>
<th>7.12-5</th>
<th>Semantics</th>
<th>In IF 2.0 processes of a system specification can be considered as an equivalent of Property of type ActiveClass.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td></td>
<td></td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface</th>
<th>7.6</th>
<th>Stereotype Interface</th>
<th>The modelling concept of an interface does not exist in IF 2.0. Nevertheless, the comparative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
</tbody>
</table>

127
<p>| &lt;&lt;Interface&gt;&gt; | 7.6-c1    | Constraint | Each nestedClassifier shall be a Signal. | This could be reused for the definition of signals for signalroutes. | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-c2    | Constraint | The ownedReception property shall be empty. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-s1    | Semantics  | A &lt;&lt;Interface&gt;&gt; Interface is mapped to an Interface-definition. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-s2    | Semantics  | The name defines the Sort of the Interface-definition. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-s3    | Semantics  | The general property defines the Data-type-identifier list that represents inheritance in the SDL abstract syntax. Each general | This constraint is not relevant for the SDL-UML to IF 2.0 transformation | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-s4 | Semantics | The nestedClassifier, ownedAttribute, ownedOperation properties define the rest of the contents of the interface. | This constraint is not relevant for the SDL-UML to IF 2.0 transformations. | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-s5 | Semantics | The ownedAttribute, ownedOperation properties are transformed to signals according to the SDL rules for remote variables (see clause 10.6 of [ITU-T Z.100]) and remote procedures (see clause 10.5 of [ITU-T Z.100]) and are thus mapped to Signals in the Signal-definition-set of the Interface-definition. | This constraint is not relevant for the SDL-UML to IF 2.0 transformations. | NA | NA | NA |
| &lt;&lt;Interface&gt;&gt; | 7.6-s6 | Semantics | Each nestedClassifier property (each of which is a Signal, see constraints above) maps to an element of the Signal-definition-set of the Interface-definition. | This could be reused for the definition of signals for signalroutes. | NA | NA | NA |
| &lt;&lt;Signal&gt;&gt; | 7.13 | Stereotype | Signal | signal-decl ::= signal-id ( [ type-id { , type-id } * ] ) | Hypothesis 5: a &lt;&lt;Signal&gt;&gt; can be transformed to Signal. | NA | NA | NA |
| &lt;&lt;Signal&gt;&gt; | 7.13-c1 | Constraint | A &lt;&lt;Signal&gt;&gt; Signal shall not have operations. | Signals in IF 2.0 do not have operations. | NA | NA | NA |
| &lt;&lt;Signal&gt;&gt; | 7.13-s1 | Semantics | A &lt;&lt;Signal&gt;&gt; Signal is mapped to a Signal-definition. The Name defines the Signal-name. The type of each ownedAttribute defines the corresponding Sort-reference-identifier. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10 | Stereotype | Port | The modelling concept of a port does not exist in IF 2.0. Nevertheless, the comparative analysis considered this SDL-UML stereotype for an alternative transformation in addition to &lt;&lt;Connector&gt;&gt;. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c1 | Constraint | The &lt;&lt;Port&gt;&gt; Port referenced by redefinedPort shall have the same name as the current Port. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c2 | Constraint | The aggregationKind shall be composite. | This constraint is not relevant for the SDL- | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c3 | Constraint | The isDerived and isDerivedUnion properties shall be false. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c4 | Constraint | The isReadOnly property shall be true. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c5 | Constraint | The defaultValue property shall be empty. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c6 | Constraint | The subsettedProperty property shall be empty. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |
| &lt;&lt;Port&gt;&gt; | 7.10-c7 | Constraint | The qualifier property shall be empty. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation s. | NA | NA | NA |</p>
<table>
<thead>
<tr>
<th>&lt;&lt;Port&gt;&gt;</th>
<th>Constraint</th>
<th>The isStatic property shall be false.</th>
<th>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</th>
<th>NA</th>
<th>NA</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>Constraint</td>
<td>The lowerValue and upperValue properties shall be ValueSpecifications that evaluate to 1.</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>Constraint</td>
<td>The isService property shall be false.</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>Semantics</td>
<td>A &lt;&lt;Port&gt;&gt; Port is mapped to a Gate-definition.</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>Semantics</td>
<td>The name defines the Gate-name.</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>Semantics</td>
<td>The requiredInterface property maps to the Out-signal-identifier-set. The set is computed according to</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>7.10-s4</td>
<td>Semantics</td>
<td>The providedInterface property defines the In-signal-identifier-set. The set is computed according to the rules given in clause 12.1.2 of [ITU-T Z.100].</td>
<td>Transformation</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Port&gt;&gt;</td>
<td>7.10-s5</td>
<td>Semantics</td>
<td>If isBehavior is true, a channel is constructed in the SDL abstract syntax that connects the gate and the state machine of the containing agent.</td>
<td>Transformation</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;SendSignalAction&gt;&gt;</td>
<td>9.20</td>
<td>Stereotype</td>
<td>SendSignalAction</td>
<td>This SDL-UML modelling concept does also exist in IF 2.0 as output.</td>
<td>Output</td>
<td>action:: == skip</td>
</tr>
<tr>
<td>&lt;&lt;SendSignalAction&gt;&gt;</td>
<td>9.20-c1</td>
<td>Constraint</td>
<td>The target property shall reference a ValuePin.</td>
<td>This is transformed to either the target process or the signalroute.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;SendSignalAction&gt;&gt;</td>
<td>9.20-c2</td>
<td>Constraint</td>
<td>The onPort property shall reference a Port of the container &lt;&lt;ActiveClass&gt;&gt; Class of the &lt;&lt;SendSignalAction&gt;&gt; SendSignalAction.</td>
<td>The concept of ports does not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;SendSignalAction&gt;&gt;</td>
<td>9.20-s1</td>
<td>Semantics</td>
<td>A &lt;&lt;SendSignalAction&gt;&gt; SendSignalAction is mapped to an Output-node. The qualifiedName of signal property maps to the Signal-identifier. The target property maps to the Signal-destination. The onPort property maps to the Direct-via. The argument property maps to the Expression list.</td>
<td>This semantics is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Activity&gt;&gt;</td>
<td>9.1</td>
<td>Stereotype</td>
<td>Activity</td>
<td>The concept of an activity is defined in IF 2.0 with the name of action.</td>
<td>NA</td>
<td>action:: = skip</td>
</tr>
<tr>
<td>&lt;&lt;Activity&gt;&gt;</td>
<td>9.1-c1</td>
<td>Constraint</td>
<td>An &lt;&lt;Activity&gt;&gt; Activity shall be empty or contain at most one ActivityNode in its node property and this node shall be a SequenceNode. An &lt;&lt;Activity&gt;&gt; Activity that is the effect of a Transition is mapped to the Graph-node list of the Transition for the</td>
<td>The concept of a sequence does not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Activity&gt;&gt;</td>
<td>9.1-c2</td>
<td>Semantics</td>
<td>Activity that is the effect of a Transition is mapped to the Graph-node list of the Transition for the</td>
<td>There is also a mapping between an activity and a transition in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Activity</td>
<td>Section</td>
<td>Semantics</td>
<td>Notes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-----------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| <<Activity>> | 9.1-c3 | An `<Activity>`
Activity that has a specification (that is, the Activity is the method of a BehavioralFeature) is mapped to a Procedure-graph containing only a Procedure-start-node consisting of a Transition. | This constraint is not relevant for the SDL-UML to IF 2.0 transformations. |
| <<Activity>> | 9.1-c4 | The actions contained in the SequenceNode map to the Graph-node list of the Transition. | SequenceNodes do not exist in IF 2.0. Actions in IF 2.0 do need to be transformed without SequenceNodes. |
| <<StateMachine>> | 8.5 | Stereotype | StateMachine | The modelling of behaviour in terms of states and transitions is specified in IF 2.0 as part of the IF 2.0 process specification. | Process state syntax: state ::= state state-id (state-option) * [tpc constraint] [save signal-id | , signal-id | "] | } (state-component) * endstate state-component ::= transition | state state-option ::= #start | #stable | #unstable | NA |
| <<StateMachine>> | 8.5-c1 Constraint | Each ownedAttribute property shall have an aggregation that is composite. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation. | NA | NA | Hypothesis 4: <<StateMachine>> can be transformed to instances of state in IF 2.0 Process. <<PseudoState>> of kind initial can be transformed to state #start. <<State>> can be transformed to state #unstable. ReceiveSignalEvent can be transformed to input. <<SendSignalAction>> can be transformed to output. |
| <<StateMachine>> | 8.5-c2 Constraint | The isReentrant property shall be false. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation. | NA | NA | NA |
| <<StateMachine>> | 8.5-c3 Constraint | If the StateMachine has a specification property, the specification property shall be an Operation. | The concept of a specification property does not exist in | NA | NA | NA |
| <<StateMachine>> | 8.5-c4 | Constraint | If the StateMachine has a specification property, the ownedParameter list of the StateMachine shall be the same as the ownedParameter list of the Operation that is the specification property. | The concept of a specification property does not exist in IF 2.0. | NA | NA | NA |
| <<StateMachine>> | 8.5-c5 | Constraint | The ownedConnector shall be empty. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation. | NA | NA | NA |
| <<StateMachine>> | 8.5-c6 | Constraint | The redefinedClassifier property shall be empty. | This constraint is not relevant for the SDL-UML to IF 2.0 transformation. | NA | NA | NA |
| <<StateMachine>> | 8.5-c7 | Constraint | If the StateMachine redefines another Behavior (as specified by redefinedBehavior), the Behavior shall be a StateMachine. | The concept of redefinedBehavior does not exist in IF 2.0. | NA | NA | NA |
| <<StateMachine>> | 8.5-c8 | Constraint | If the StateMachine redefines another StateMachine (as specified by redefinedBehavior, or extendedStateMachine), it shall have the same. | The concept of redefinedBehavior does not exist in IF 2.0. | NA | NA | NA |
name as the redefined StateMachine.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
<th>8.5-c9 Constraint</th>
<th>8.5-c10 Constraint</th>
<th>8.5-c11 Constraint</th>
<th>8.5-c12 Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the StateMachine is the classifierBehavior of a Class, the redefinedBehavior property shall be empty.</td>
<td>The concept of redefinedBehavior does not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>If the StateMachine is not the classifierBehavior of a Class, then the extendedStateMachine property shall be empty.</td>
<td>The concept of redefinedBehavior does not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>If a StateMachine is mapped to a Composite-state-type (see the Semantics clause below): The returnedResult property shall be empty (so that StateMachine does not return a result).</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>If a StateMachine is mapped to a Procedure-graph (see the Semantics clause below): There shall only be one Region. The connectionPoint property shall be empty. The classifierBehavior shall be empty. The ownedPort shall be</td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>
empty. The general property shall be empty.

<table>
<thead>
<tr>
<th>&lt;&lt;StateMachine&gt;&gt;</th>
<th>8.5-c13</th>
<th>Constraint</th>
<th>Interfaces do not exist in IF 2.0.</th>
<th>NA</th>
<th>NA</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;StateMachine&gt;&gt;</td>
<td>8.5-s1</td>
<td>Semantics</td>
<td>None of the semantics of the mapping of SDL-UML to SDL state machines is relevant for the transformation of the SDL-UML ticket machine model state machines. Therefore, the semantics will not be reproduced in this table.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>stereotype</td>
<td>8.4</td>
<td>constraint</td>
<td>states also exist in if 2.0.</td>
<td>NA</td>
<td>syntax: state ::= state state-state-id ( state-option ) * [ tpc constraint ] [ save signal-id , signal-id ] . state-component ::= state-component * endstate state-component ::= transition</td>
<td>state state-option ::= #start</td>
</tr>
<tr>
<td>stereotype</td>
<td>8.4-c1</td>
<td>constraint</td>
<td>The doActivity property shall be empty.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>stereotype</td>
<td>8.4-c2</td>
<td>constraint</td>
<td>The entry and exit properties shall be empty, because entry/exit actions are not supported.</td>
<td>Entry and exit properties do not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
### Constraint 8.4-c3

**The isComposite property shall be false, because only decomposition using submachine properties is allowed and a State shall have an empty region property.**

There is no decomposition of states in IF 2.0.

| NA | NA | NA |

### Constraint 8.4-c4

**In the Transition set defined by the outgoing properties of a State, the signal property of each event property that is a SignalEvent of each trigger shall be distinct.**

There is no such constraint in the IF 2.0 language definition.

| NA | NA | NA |

### Stereotype PseudoState

**This concept also exists in the IF 2.0 language specification.**

Hypothesis 4: <<StateMachine>> can be transformed to instances of state in IF 2.0

- <<StateMachine>> of kind initial can be transformed to state #start,
- <<StateMachine>> of kind choice can be transformed to state #unstable,
- ReceiveSignalEvent can be transformed to state #unstable.

| state #start | state-option ::= #start | Hypothesis 4: <<StateMachine>> can be transformed to instances of state in IF 2.0
- <<StateMachine>> of kind initial can be transformed to state #start,
- <<StateMachine>> of kind choice can be transformed to state #unstable,
- ReceiveSignalEvent can be transformed to state #unstable. |
<table>
<thead>
<tr>
<th>&lt;&lt;PseudoState&gt;&gt;</th>
<th>8.2-c1</th>
<th>Constraint</th>
<th>A Transition shall have an empty guard property if the Transition is an outgoing property of a &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == initial.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>In IF 2.0, a transition of state-option ::= #start can also have a guard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>transition ::= [deadline {eager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;&lt;PseudoState&gt;&gt;</th>
<th>8.2-c2</th>
<th>Constraint</th>
<th>A Transition shall have an empty trigger property if the Transition is an outgoing property of a &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == initial.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>In IF 2.0, a transition of state-option ::= #start can also have a trigger.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;&lt;PseudoState&gt;&gt;</th>
<th>8.2-c3</th>
<th>Constraint</th>
<th>The classifierBehavior of a non-abstract &lt;&lt;ActiveClass&gt;&gt; Class shall have an &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == initial.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>This constraint is not relevant for the SDL-UML to IF 2.0 transformation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-c4</td>
<td>Constraint</td>
<td>The kind property of &lt;&lt;PseudoState&gt;&gt; shall not be join or fork or shallowHistory.</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-c5</td>
<td>Constraint</td>
<td>A &lt;&lt;PseudoState&gt;&gt; with kind == deepHistory or with kind == exitPoint or with kind == terminate shall not have an outgoing property.</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-c6</td>
<td>Constraint</td>
<td>A Transition shall have a non-empty guard property Constraint (a Boolean Expression) if the Transition is an outgoing property of a &lt;&lt;PseudoState&gt;&gt; with kind == choice.</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-c7</td>
<td>Constraint</td>
<td>Each guard of each Transition that is an outgoing property of a &lt;&lt;PseudoState&gt;&gt; with kind == choice shall be an Expression with two operand properties. One operand shall be identical in every such guard of the &lt;&lt;PseudoState&gt;&gt;. For the purposes of description, such an operand is called the left-hand operand.</td>
</tr>
</tbody>
</table>
A description of the other operand is called the right-hand operand, and shall evaluate to a value set (possibly with just one element) with elements of the same data type as the left-hand operand. The value set defined by a right-hand operand shall be statically determinable.

<table>
<thead>
<tr>
<th>&lt;&lt;PseudoState&gt;&gt;</th>
<th>Semantics</th>
<th>8.2-s1</th>
<th>8.2-s2</th>
<th>8.2-s3</th>
<th>8.2-s4</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == initial is mapped to a Procedure-start-node in a region that defines a Procedure-graph and State-start-node in a region that defines a CompositeState-graph.</td>
<td>This can be mapped to state-option ::= #start.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == deepHistory is mapped to a NextState-node that is a dash-nextstate with HISTORY.</td>
<td>This kind of state does not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == junction is mapped to a Free-action and one or more</td>
<td>This kind of state does not exist in IF 2.0.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == choice is mapped to a Decision-node.</td>
<td>This can be mapped to state-option ::= #unstable.</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-s5</td>
<td>Semantics</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == entryPoint is mapped to a Start-state-node.</td>
<td>This kind of state does not exist in IF 2.0.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-s6</td>
<td>Semantics</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == exitPoint is mapped to a Named-return node.</td>
<td>This kind of state does not exist in IF 2.0.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;PseudoState&gt;&gt;</td>
<td>8.2-s7</td>
<td>Semantics</td>
<td>A &lt;&lt;PseudoState&gt;&gt; PseudoState with kind == terminate is mapped to a Stop-node.</td>
<td>This kind of state does not exist in IF 2.0.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Transition&gt;&gt;</td>
<td>8.6</td>
<td>Stereotype</td>
<td>Transition</td>
<td>This concept also exists in the IF 2.0 language specification.</td>
<td>Transition transition ::= [deadline {eager</td>
</tr>
<tr>
<td>&lt;&lt;Transition&gt;&gt;</td>
<td>8.6-c1</td>
<td>Constraint</td>
<td>The Transition shall have kind == external or local. The UML concept of internal transitions is not allowed.</td>
<td>This concept does not exist in the IF 2.0 language specification.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Transition&gt;&gt;</td>
<td>8.6-c2</td>
<td>Constraint</td>
<td>The trigger property shall not be empty.</td>
<td>This constraint does not exist in the IF 2.0 language specification.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Transition&gt;&gt;</td>
<td>8.6-c3</td>
<td>Constraint</td>
<td>The port of the Trigger that is the trigger property of the Transition shall be empty.</td>
<td>The concept of ports does not exist in IF 2.0.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Transition&gt;&gt;</td>
<td>8.6-c4</td>
<td>Constraint</td>
<td>The event property of the trigger property shall be a MessageEvent or ChangeEvent.</td>
<td>In IF 2.0 this can be the receipt of a signal.</td>
<td>NA</td>
</tr>
<tr>
<td>&lt;&lt;Transition&gt;&gt;</td>
<td>8.6-c5</td>
<td>Constraint</td>
<td>The effect property shall reference an Activity.</td>
<td>This is also the case in the IF 2.0 language specification.</td>
<td>NA</td>
</tr>
</tbody>
</table>
Appendix D - SDL-UML ticket machine model
Appendix E - IF 2.0 ticket machine example with explicitly defined signalroutes

system TicketMachine;

signal EmptyTicketTray1(boolean);
signal EmptyTicketTray2(boolean);

signalroute sig0(1) from env to TicketMachine with EmptyTicketTray1;
signalroute sig1(1) from TicketMachine to env with EmptyTicketTray1;
signalroute sig2(1) from TicketMachine to TicketTray with EmptyTicketTray2;
signalroute sig3(1) from TicketTray to TicketMachine with EmptyTicketTray2;

process TicketMachine(1);

var success boolean;
var resultEmptyTicketTray boolean;

state start1 #start;
  nextstate initialized;
  endstate;

state initialized;
input EmptyTicketTray1(resultEmptyTicketTray);
output EmptyTicketTray2(resultEmptyTicketTray) via {sig2}0;
  nextstate call_TicketTray;
  endstate;

state call_TicketTray;
input EmptyTicketTray2(resultEmptyTicketTray);
task success := resultEmptyTicketTray;
output EmptyTicketTray1(success) via {sig2}0;
nextstate initialized;
endstate;
endprocess;

process TicketTray(1);

state ready #init;
input EmtpyTicketTray1();
output EmtpyTicketTray2(false) via {sig3}0;
nexstate ready;
endstate;
endprocess;

endsystem;
Appendix F – AGEDIS IF 2.0 ticket machine specification

/******************************************************************************
**************
IF model generated by AGEDIS AML compiler. See www.agedis.de.
Aml compiler version: 1.26
*******************************************************************************/

system IF_ModelGeneratedFromAml;

type TicketsInStore = range 0 .. 4;

type CoinsInStore = range 0 .. 7;

type CoinsInTray = range 0 .. 4;

signal IF_call_CoinBuffer_cbReceiveCoin (pid);

signal IF_return_TicketStore_tsPrintTicket (boolean);

signal IF_call_TicketStore_tsPrintTicket (pid);

signal IF_return_TicketStore_tsIsReadyToPrint (boolean);

signal IF_call_TicketStore_tsIsReadyToPrint (pid);

signal IF_return_Control_cReturnCoins (boolean);

signal IF_call_TicketMachine_removeTicket ();

signal IF_return_Control_cRequestTicket (boolean);
signal IF_call_TicketMachine_removeCoins ();

signal IF_call_TicketMachine_returnCoins ();

signal IF_call_TicketMachine_printTicket ();

signal IF_call_TicketMachine_insertCoin ();

signal IF_DoNothing ();

signal IF_return_TicketTray_ttEmptyTray (boolean);

signal IF_call_TicketTray_ttEmptyTray (pid);

signal IF_return_Control_cInsertCoin (boolean);

signal IF_return_TicketTray_ttReceiveTicket (boolean);

signal IF_call_TicketTray_ttReceiveTicket (pid);

signal IF_call_Control_cReturnCoins (pid);

signal IF_return_CoinStore_csReceiveCoinsFromBuffer (boolean);

signal IF_call_Control_cRequestTicket (pid);

signal IF_call_CoinStore_csReceiveCoinsFromBuffer (pid);

signal IF_return_CoinTray_ctReceiveCoins (boolean);

signal IF_call_CoinTray_ctReceiveCoins (integer, pid);

signal IF_return_CoinBuffer_cbCoinsToTray (boolean);
signal IF_call_CoinBuffer_cbCoinsToTray (pid);

signal IF_return_CoinBuffer_cbCoinsToStore (boolean);

signal IF_call_CoinBuffer_cbCoinsToStore (pid);

signal IF_return_TicketMachine_removeTicket ()

signal IF_return_TicketMachine_removeCoins ()

signal IF_return_TicketMachine_returnCoins (boolean);

signal IF_return_TicketMachine_printTicket (boolean);

signal IF_return_TicketMachine_insertCoin (boolean);

signal IF_return_TicketTray_ttIsEmpty (boolean);

signal IF_call_TicketTray_ttIsEmpty (pid);

signal IF_return_CoinTray_ctEmptyCoinTray (boolean);

signal IF_call_CoinTray_ctEmptyCoinTray (pid);

signal IF_call_Control_cInsertCoin (pid);

signal IF_return_CoinBuffer_cbReceiveCoin (boolean);

const COINSTORE_CAP = 6;

const COINTRAY_CAP = 3;

const TICKETTRAY_CAP = 1;
const TICKETSTORE_CAP = 3;
const COINBUFFER_CAP = 3;
const TICKET_PRICE = 2;
const INIT_TICKETS_IN_STORE = 3;

signalroute IF_env_sigroute_0(1)
  from env to CoinBuffer
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
signalroute IF_env_sigroute_1(1)
  from env to CoinStore
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
signalroute IF_env_sigroute_2(1)
  from env to TicketMachine
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
signalroute IF_env_sigroute_3(1)
  from env to Control
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
signalroute IF_env_sigroute_4(1)
  from env to TicketTray
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
signalroute IF_env_sigroute_5(1)
  from env to CoinTray
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
signalroute IF_env_sigroute_6(1)
  from env to TicketStore
  with IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_DoNothing;
process CoinBuffer (1);

var IF_result_TicketStore_tsIsReadyToPrint boolean;
var IF_result_CoinTray_ctEmptyCoinTray boolean;
var IF_result_TicketMachine_returnCoins boolean;
var IF_result_TicketMachine_printTicket boolean;
var IF_result_TicketMachine_insertCoin boolean;
var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_result_Control_cReturnCoins boolean;
var storeHasAccepted boolean;
var IF_tmp_target pid;
var IF_result_Control_cRequestTicket boolean;
var IF_sender pid;
var IF_result_TicketTray_ttIsEmpty boolean;
var trayHasAccepted boolean;
var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;
var IF_result_TicketTray_ttEmptyTray boolean;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;
var IF_result_Control_cInsertCoin boolean;
var coinsInBuffer integer;
var IF_call_sender pid;
var IF_result_CoinBuffer_cbCoinsToStore boolean;

type IF_ConnectivityTableLine_control_0 = array [1] of pid;
type IF_ConnectivityTable_control_0 = array [1] of IF_ConnectivityTableLine_control_0;
var IF_Connectivity_control IF_ConnectivityTable_control_0;

procedure IF_Init_control;
procedure IF_control;
  fpar in x IF_ConnectivityTable_control_0, in sourceProcessId integer, in
  targetProcessIndex integer;
  returns pid;
  {#
    return x [sourceProcessId] [targetProcessIndex];
  #}
endprocedure;

type IF_ConnectivityTableLine_coinTray_0 = array [1] of pid;
type IF_ConnectivityTable_coinTray_0 = array [1] of
  IF_ConnectivityTableLine_coinTray_0;
var IF_Connectivity_coinTray IF_ConnectivityTable_coinTray_0;

procedure IF_Init_coinTray;
  fpar out x IF_ConnectivityTable_coinTray_0;
  {#
    x [0] [0] = if_pid_mk(if_CoinTray_process, 0);
  #}
endprocedure;

procedure IF_coinTray;
  fpar in x IF_ConnectivityTable_coinTray_0, in sourceProcessId integer, in
  targetProcessIndex integer;
  returns pid;
  {#
    return x [sourceProcessId] [targetProcessIndex];
  #}
endprocedure;
type IF_ConnectivityTableLine_coinStore_0 = array [1] of pid;

type IF_ConnectivityTable_coinStore_0 = array [1] of IF_ConnectivityTableLine_coinStore_0;

var IF_Connectivity_coinStore IF_ConnectivityTable_coinStore_0;

procedure IF_Init_coinStore;
  fpar out x IF_ConnectivityTable_coinStore_0;
  
  {#
    x [0] [0] = if_pid_mk(if_CoinStore_process, 0);
  #}
endprocedure;

procedure IF_coinStore;
  fpar in x IF_ConnectivityTable_coinStore_0, in sourceProcessId integer, in targetProcessIndex integer;
  returns pid;
  
  {#
    return x [sourceProcessId] [targetProcessIndex];
  #}
endprocedure;

state NoNameTag_14 #start ;


call IF_Init_coinStore(IF_Connectivity_coinStore);
call IF_Init_coinTray(IF_Connectivity_coinTray);
call IF_Init_control(IF_Connectivity_control);
task IF_sender := nil;
task coinsInBuffer := 0;
nextstate coinBufferNotFull;
endstate;

state coinBufferNotFull ;
input IF_call_CoinBuffer_cbCoinsToTray(IF_sender);

task IF_call_sender := IF_sender;
IF_tmp_target := call IF_coinTray(IF_Connectivity_coinTray, {integer} self, 0);
output IF_call_CoinTray_ctReceiveCoins (coinsInBuffer, self) to IF_tmp_target;

task IF_sender := nil;
nextstate IF_state_IF_call_42;

input IF_call_CoinBuffer_cbReceiveCoin(IF_sender);

task IF_call_sender := IF_sender;
task coinsInBuffer := coinsInBuffer + 1;
task IF_sender := nil;
nextstate IF_branch_state_2;

input IF_call_CoinBuffer_cbCoinsToStore(IF_sender);

task IF_call_sender := IF_sender;
task IF_sender := nil;
nextstate IF_branch_state_6;
endstate;

state IF_branch_state_2 #unstable ;
provided coinsInBuffer < COINBUFFER_CAP ;

task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbReceiveCoin (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinBufferNotFull;
provided coinsInBuffer >= COINBUFFER_CAP ;

task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbReceiveCoin (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinBufferFull;
endstate;

state IF_state_IF_call_42 ;

save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_TicketStore_tsPrintTicket, IF_return_Control_cReturnCoins,
IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_return_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_return_Control_cInsertCoin, IF_return_TicketTray_ttReceiveTicket,
IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
IF_return_CoinStore_csReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
IF_call_CoinStore_csReceiveCoinsFromBuffer, IF_call_CoinTray_ctReceiveCoins,
IF_return_CoinBuffer_cbCoinsToTray, IF_call_CoinBuffer_cbCoinsToTray,
IF_return_CoinBuffer_cbCoinsToStore, IF_call_CoinBuffer_cbCoinsToStore,
IF_return_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
IF_return_TicketMachine_returnCoins, IF_return_TicketMachine_printTicket,
IF_return_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
IF_call_CoinTray_ctEmptyCoinTray, IF_call_TicketTray_ttIsEmpty,
IF_call_CoinTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
IF_return_CoinBuffer_cbReceiveCoin;

input IF_return_CoinTray_ctReceiveCoins(IF_result_CoinTray_ctReceiveCoins);

task trayHasAccepted := IF_result_CoinTray_ctReceiveCoins;

nextstate IF_branch_state_3;
endstate;

state IF_branch_state_4 #unstable ;

provided trayHasAccepted = true ;
task coinsInBuffer := 0;

task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbCoinsToTray (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinBufferNotFull;
provided trayHasAccepted = false ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbCoinsToTray (false) to IF_tmp_target;
task IF_sender := nil;
nextstate coinBufferFull;
endstate;
state IF_state_IF_call_39 ;
  save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_call_TicketStore_tsPrintTicket, IF_return_Control_cReturnCoins,
  IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
  IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
  IF_return_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
  IF_return_Control_cInsertCoin, IF_return_TicketTray_ttReceiveTicket,
  IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
  IF_return_CoinStore_csReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
  IF_call_CoinStore_csReceiveCoinsFromBuffer, IF_call_CoinTray_ctReceiveCoins,
  IF_call_TicketMachine_removeTicket, IF_call_CoinBuffer_cbCoinsToTray,
  IF_call_CoinBuffer_cbCoinsToStore, IF_call_CoinBuffer_cbCoinsToTray,
  IF_return_TicketMachine_removeCoin, IF_call_TicketMachine_returnCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
  IF_return_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
  IF_call_CoinTray_ctEmptyCoinTray, IF_call_Control_cInsertCoin,
  IF_return_CoinBuffer_cbReceiveCoin;
  input IF_result_CoinTray_ctReceiveCoins(IF_result_CoinTray_ctReceiveCoins);
  task trayHasAccepted := IF_result_CoinTray_ctReceiveCoins;
  nextstate IF_branch_state_4;
endstate;
state IF_branch_state_6 #unstable ;
  provided coinsInBuffer < TICKET_PRICE ;
  task IF_sender := IF_call_sender;
  task IF_tmp_target := IF_sender;
  output IF_result_CoinBuffer_cbCoinsToStore (false) to IF_tmp_target;
  task IF_sender := nil;
  nextstate coinBufferNotFull;
  provided coinsInBuffer >= TICKET_PRICE ;
  IF_tmp_target := call IF_coinStore(IF_Connectivity_coinStore, {integer}
  self, 0);;
  output IF_result_CoinStore_csReceiveCoinsFromBuffer (self) to IF_tmp_target;
  task IF_sender := nil;
  nextstate IF_state_IF_call_36;
endstate;
state IF_state_IF_call_36 ;
  save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_call_TicketStore_tsPrintTicket, IF_return_Control_cReturnCoins,
  IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,

input
IF_return_CoinStore_csReceiveCoinsFromBuffer (IF_result_CoinStore_csReceiveCoinsFromBuffer);

task storeHasAccepted := IF_result_CoinStore_csReceiveCoinsFromBuffer;

if(storeHasAccepted)then

task coinsInBuffer := coinsInBuffer - TICKET_PRICE;

endif

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_CoinBuffer_cbCoinsToStore (storeHasAccepted) to IF_tmp_target;

task IF_sender := nil;

nextstate coinBufferNotFull;
endstate;

state IF_state_IF_call_35 ;

IF_return_TicketMachine_returnCoins, IF_return_TicketMachine_printTicket, 
IF_return_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty, 
IF_return_CoinTray_ctEmptyCoinTray, IF_call_TicketTray_ttIsEmpty, 
IF_call_CoinTray_ctEmptyCoinTray, IF_call_Control_cInsertCoin, 
IF_return_CoinBuffer_cbReceiveCoin;

input 
IF_return_CoinStore_csReceiveCoinsFromBuffer(IF_result_CoinStore_csReceiveCoinsFromBuffer);

nextstate IF_branch_state_5;
endstate;

state IF_branch_state_5 #unstable ;
provided storeHasAccepted = true ;
task coinsInBuffer := coinsInBuffer -TICKET_PRICE;

nexttask IF_sender := IF_call_sender;
nexttask IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbCoinsToStore (true) to IF_tmp_target;
nexttask IF_sender := nil;
nextstate coinBufferNotFull;
provided storeHasAccepted = false ;
task IF_sender := IF_call_sender;
nexttask IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbCoinsToTray (false) to IF_tmp_target;
nexttask IF_sender := nil;
nextstate coinBufferFull;
endstate;

state IF_branch_state_3 #unstable ;
provided trayHasAccepted = true ;
task coinsInBuffer := 0;

nexttask IF_sender := IF_call_sender;
nexttask IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbCoinsToTray (true) to IF_tmp_target;
nexttask IF_sender := nil;
nextstate coinBufferNotFull;
provided trayHasAccepted = false ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbCoinsToTray (false) to IF_tmp_target;
task IF_sender := nil;
nextstate coinBufferNotFull;
endstate;

state coinBufferFull ;
input IF_call_CoinBuffer_cbReceiveCoin(IF_sender);
task IF_call_sender := IF_sender;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinBuffer_cbReceiveCoin (false) to IF_tmp_target;
task IF_sender := nil;
nextstate coinBufferFull;
input IF_call_CoinBuffer_cbCoinsToTray(IF_sender);
task IF_call_sender := IF_sender;
IF_tmp_target := call IF_coinTray(IF_Connectivity_coinTray, {integer} self, 0);
output IF_call_CoinTray_ctReceiveCoins (coinsInBuffer, self) to IF_tmp_target;
task IF_sender := nil;
nextstate IF_state_IF_call_39;
input IF_call_CoinBuffer_cbCoinsToStore(IF_sender);
task IF_call_sender := IF_sender;
IF_tmp_target := call IF_coinStore(IF_Connectivity_coinStore, {integer} self, 0);
output IF_call_CoinStore_csReceiveCoinsFromBuffer (self) to IF_tmp_target;
task IF_sender := nil;
nextstate IF_state_IF_call_35;
endstate;
endprocess;
process CoinStore (1);

var IF_result_Control_cInsertCoin boolean;
var IF_result_CoinBuffer_cbCoinsToStore boolean;
var IF_result_TicketStore_tsIsReadyToPrint boolean;
var IF_result_CoinTray_ctEmptyCoinTray boolean;
var IF_tmp_target pid;
var IF_sender pid;
var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_result_TicketMachine_returnCoins boolean;
var coinsInStore CoinsInStore;
var IF_result_TicketMachine_printTicket boolean;
var IF_result_TicketMachine_insertCoin boolean;
var IF_result_Control_cReturnCoins boolean;
var IF_result_TicketTray_ttIsEmpty boolean;
var IF_result_Control_cRequestTicket boolean;
var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;
var IF_result_TicketTray_ttEmptyTray boolean;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;
var IF_call_sender pid;

type IF_ConnectivityTableLine_coinBuffer_1 = array [1] of pid;
type IF_ConnectivityTable_coinBuffer_1 = array [1] of
  IF_ConnectivityTableLine_coinBuffer_1;
var IF_Connectivity_coinBuffer IF_ConnectivityTable_coinBuffer_1;

procedure IF_Init_coinBuffer;
  fpar out x IF_ConnectivityTable_coinBuffer_1;
  {#
procedure IF_coinBuffer;

fpar in x IF_ConnectivityTable_coinBuffer_1, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

state NoNameTag_26 #start ;


call IF_Init_coinBuffer(IF_Connectivity_coinBuffer);

task IF_sender := nil;

task coinsInStore := 0;

nextstate coinStoreWaiting;
endstate;

state coinStoreWaiting ;

input IF_call_CoinStore_csReceiveCoinsFromBuffer(IF_sender);

task IF_call_sender := IF_sender;

task coinsInStore := coinsInStore + TICKET_PRICE;

task IF_sender := nil;
nextstate IF_branch_state_9;
endstate;

state coinStoreFull ;
input IF_call_CoinStore_csReceiveCoinsFromBuffer(IF_sender);
task IF_call_sender := IF_sender;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinStore_csReceiveCoinsFromBuffer (false) to IF_tmp_target;
task IF_sender := nil;
nextstate coinStoreFull;
endstate;

state IF_branch_state_9 #unstable ;
provided coinsInStore < COINSTORE_CAP ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinStore_csReceiveCoinsFromBuffer (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinStoreWaiting;
provided coinsInStore >= COINSTORE_CAP ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinStore_csReceiveCoinsFromBuffer (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinStoreFull;
endstate;
endprocess;

process TicketMachine (1);

var IF_result_TicketTray_ttEmptyTray boolean;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var IF_result_TicketMachine_returnCoins boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;
var IF_result_TicketMachine_printTicket boolean;
var IF_result_TicketMachine_insertCoin boolean;
var success boolean;
var IF_tmp_target pid;
var IF_result_Control_cReturnCoins boolean;
var IF_sender pid;
var IF_result_Control_cRequestTicket boolean;
var IF_result_CoinBuffer_cbCoinsToStore boolean;
var IF_result_TicketStore_tsIsReadyToPrint boolean;
var IF_result_CoinTray_ctEmptyCoinTray boolean;
var IF_SlowEnvironmentClock_0 clock;
var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_result_Control_cInsertCoin boolean;
var IF_result_TicketTray_ttIsEmpty boolean;
var IF_call_sender pid;
var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;

type IF_ConnectivityTableLine_ticketTray_2 = array [1] of pid;
type IF_ConnectivityTable_ticketTray_2 = array [1] of IF_ConnectivityTableLine_ticketTray_2;
var IF_Connectivity_ticketTray IF_ConnectivityTable_ticketTray_2;

procedure IF_Init_ticketTray;
fpar out x IF_ConnectivityTable_ticketTray_2;
{#
 x [0] [0] = if_pid_mk(if_TicketTray_process, 0);
 #}
endprocedure;
procedure IF_ticketTray;
    fpar in x IF_ConnectivityTable_ticketTray_2, in sourceProcessId integer, in targetProcessIndex integer;
    returns pid;
    {
        return x [sourceProcessId] [targetProcessIndex];
    }
endprocedure;

type IF_ConnectivityTableLine_control_2 = array [1] of pid;

type IF_ConnectivityTable_control_2 = array [1] of IF_ConnectivityTableLine_control_2;

var IF_Connectivity_control IF_ConnectivityTable_control_2;

procedure IF_Init_control;
    fpar out x IF_ConnectivityTable_control_2;
    {
        x [0] [0] = if_pid_mk(if_Control_process, 0);
    }
endprocedure;

procedure IF_control;
    fpar in x IF_ConnectivityTable_control_2, in sourceProcessId integer, in targetProcessIndex integer;
    returns pid;
    {
        return x [sourceProcessId] [targetProcessIndex];
    }
endprocedure;

procedure IF_init_coinTray;
    fpar in x IF_ConnectivityTable_coinTray_2, in sourceProcessId integer, in targetProcessIndex integer;
    returns pid;
    {
        return x [sourceProcessId] [targetProcessIndex];
    }
endprocedure;

procedure IF_init_ticketTray;
    fpar in x IF_ConnectivityTable_ticketTray_2, in sourceProcessId integer, in targetProcessIndex integer;
    returns pid;
    {
        return x [sourceProcessId] [targetProcessIndex];
    }
endprocedure;
procedure IF_Init_coinTray;
  fpar out x IF_ConnectivityTable_coinTray_2;
  {#
    x [0] [0] = if_pid_mk(if_CoinTray_process, 0);
  #}
endprocedure;

procedure IF_coinTray;
  fpar in x IF_ConnectivityTable_coinTray_2, in sourceProcessId integer, in
  targetProcessIndex integer;
  returns pid;
  {#
    return x [sourceProcessId] [targetProcessIndex];
  #}
endprocedure;

state NoNameTag_42 #start ;
  save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore-tsPrintTicket,
  IF_call_TicketStore-tsPrintTicket, IF_return_TicketStore-IsReadyToPrint,
  IF_call_TicketMachine-IsReadyToPrint, IF_return_Control-cReturnCoins,
  IF_call_TicketMachine-removeTicket, IF_return_Control-cRequestTicket,
  IF_call_TicketMachine-removeCoins, IF_call_TicketMachine-returnCoins,
  IF_call_TicketMachine-printTicket, IF_call_TicketMachine-insertCoin, IF_DoNothing,
  IF_return_TicketTray-IsEmptyTray, IF_call_TicketTray-IsEmptyTray,
  IF_return_Control-cInsertCoin, IF_return_TicketTray-receiveTicket,
  IF_call_TicketTray-receiveTicket, IF_call_Control-cReturnTickets,
  IF_return_CoinStore-receiveCoinsFromBuffer, IF_call_Control-cRequestTicket,
  IF_call_CoinStore-receiveCoinsFromBuffer, IF_return_CoinTray-receiveCoins,
  IF_call_CoinBuffer-receiveCoins, IF_return_CoinBuffer-cRequestTicket,
  IF_call_CoinBuffer-cRequestTicket, IF_return_CoinBuffer-cRequestTicket,
  IF_call_TicketMachine-receiveTickets, IF_return_TicketMachine-insertCoin,
  IF_return_TicketTray-IsEmpty, IF_return_CoinTray-IsEmptyCoinTray,
  IF_call_TicketTray-IsEmpty, IF_call_CoinTray-IsEmptyCoinTray,
  IF_call_Control-cInsertCoin, IF_return_CoinBuffer-receiveCoin;

call IF_Init_coinTray(IF_Connectivity_coinTray);
  call IF_Init_control(IF_Connectivity_control);
  call IF_Init_ticketTray(IF_Connectivity_ticketTray);
  set IF_SlowEnvironmentClock_0 := 0;
  task IF_sender := nil;
  nextstate initialized;
endstate;
state IF_state_IF_call_45;

  save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_call_TicketStore_tsPrintTicket, IF_return_TicketStore_tsiIsReadyToPrint,
  IF_call_TicketMachine_removeTicket, IF_return_Control_cReturnCoins,
  IF_call_TicketMachine_removeCoin, IF_call_TicketMachine_printTicket,
  IF_call_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_call_TicketTray_ttReceiveTicket, IF_return_Control_cInsertCoin,
  IF_return_TicketMachine_removeTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_return_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cInsertCoin,
  IF_return_TicketMachine_removeTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_return_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cInsertCoin,
  IF_return_TicketMachine_removeTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_return_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cInsertCoin,
  IF_return_TicketMachine_removeTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_return_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cInsertCoin,
  IF_return_TicketMachine_removeTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_return_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cInsertCoin,
  IF_return_TicketMachine_removeTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
input IF_DoNothing();

set IF_SlowEnvironmentClock_0 := 0;

nextstate IF_state_IF_call_44;

input IF_return_CoinTray_ctEmptyCoinTray(IF_result_CoinTray_ctEmptyCoinTray);

task success := IF_result_CoinTray_ctEmptyCoinTray;

output IF_return_TicketMachine_removeCoins();

nextstate initialized;
endstate;

state IF_state_IF_call_43;

save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_TicketStore_tsIsReadyToPrint, IF_return_Control_cReturnCoins,
IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin;

when IF_SlowEnvironmentClock_0 = 1;

input IF_DoNothing();

set IF_SlowEnvironmentClock_0 := 0;

nextstate IF_state_IF_call_43;

input IF_return_TicketTray_ttEmptyTray(IF_result_TicketTray_ttEmptyTray);

task success := IF_result_TicketTray_ttEmptyTray;

output IF_return_TicketMachine_removeTicket();

nextstate initialized;
endstate;
state IF_state_IF_call_40;


input IF_return_Control_cReturnCoins(IF_result_Control_cReturnCoins);

task success := IF_result_Control_cReturnCoins;

output IF_return_TicketMachine_returnCoins {success};

nextstate initialized;

when IF_SlowEnvironmentClock_0 = 1;

input IF_DoNothing();

set IF_SlowEnvironmentClock_0 := 0;

nextstate IF_state_IF_call_40;
endstate;

state initialized;

when IF_SlowEnvironmentClock_0 = 1;

input IF_DoNothing();

set IF_SlowEnvironmentClock_0 := 0;

nextstate initialized;

when IF_SlowEnvironmentClock_0 = 1;

input IF_call_TicketMachine_returnCoins();

set IF_SlowEnvironmentClock_0 := 0;

task IF_call_sender := IF_sender;

IF_tmp_target := call IF_control(IF_Connectivity_control, {integer} self, 0);
output IF_call_Control_cReturnCoins (self) to IF_tmp_target;
nextstate IF_state_IF_call_40;
when IF_SlowEnvironmentClock_0 = 1;
input IF_call_TicketMachine_removeTicket();
set IF_SlowEnvironmentClock_0 := 0;
task IF_call_sender := IF_sender;
IF_tmp_target := call IF_ticketTray(IF_Connectivity_ticketTray, {integer} self, 0);
output IF_call_TicketTray_ttEmptyTray (self) to IF_tmp_target;
nextstate IF_state_IF_call_43;
when IF_SlowEnvironmentClock_0 = 1;
input IF_call_TicketMachine_insertCoin();
set IF_SlowEnvironmentClock_0 := 0;
task IF_call_sender := IF_sender;
IF_tmp_target := call IF_control(IF_Connectivity_control, {integer} self, 0);
output IF_call_Control_cInsertCoin (self) to IF_tmp_target;
nextstate IF_state_IF_call_37;
when IF_SlowEnvironmentClock_0 = 1;
input IF_call_TicketMachine_removeCoins();
set IF_SlowEnvironmentClock_0 := 0;
task IF_call_sender := IF_sender;
IF_tmp_target := call IF_coinTray(IF_Connectivity_coinTray, {integer} self, 0);
output IF_call_CoinTray_ctEmptyCoinTray (self) to IF_tmp_target;
nextstate IF_state_IF_call_44;
when IF_SlowEnvironmentClock_0 = 1;
input IF_call_TicketMachine_printTicket();
set IF_SlowEnvironmentClock_0 := 0;
task IF_call_sender := IF_sender;
IF_tmp_target := call IF_control(IF_Connectivity_control, {integer} self, 0);
output IF_call_Control_cRequestTicket (self) to IF_tmp_target;
nextstate IF_state_IF_call_45;
endstate;
state IF_state_IF_call_37 ;

  save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
  IF_call_TicketStore_tsPrintTicket, IF_return_TicketStore_tsIsReadyToPrint,
  IF_call_TicketStore_tsIsReadyToPrint, IF_return_Control_cReturnCoins,
  IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
  IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
  IF_return_TicketMachine_removeTicket, IF_call_TicketMachine_removeCoins,
  IF_call_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
  IF_return_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
  IF_return_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
  IF_return_TicketTray_ttReceiveTicket, IF_call_TicketTray_ttReceiveTicket,
  IF_call_Control_cReturnCoins, IF_return_CoinStore_csReceiveCoinsFromBuffer,
  IF_call_Control_cRequestTicket, IF_call_CoinStore_csReceiveCoinsFromBuffer,
  IF_return_CoinTray_ctReceiveCoins, IF_call_CoinTray_ctReceiveCoins,
  IF_return_CoinBuffer_cbCoinsToTray, IF_call_CoinBuffer_cbCoinsToTray,
  IF_return_CoinBuffer_cbCoinsToStore, IF_call_CoinBuffer_cbCoinsToStore,
  IF_return_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
  IF_return_TicketMachine_returnCoins, IF_return_TicketMachine_printTicket,
  IF_return_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
  IF_return_CoinTray_ctEmptyCoinTray, IF_call_CoinTray_ctEmptyCoinTray,
  IF_call_CoinTray_ctEmptyCoinTray, IF_call_Control_cInsertCoin,
  IF_return_CoinBuffer_cbReceiveCoin;

  when IF_SlowEnvironmentClock_0 = 1 ;
  input IF_DoNothing();

  set IF_SlowEnvironmentClock_0 := 0;

  nextstate IF_state_IF_call_37;

  input IF_return_Control_cInsertCoin(IF_result_Control_cInsertCoin);

  task success := IF_result_Control_cInsertCoin;

  output IF_return_TicketMachine_insertCoin (success);

  nextstate initialized;

  endstate;

endprocess;

process Control (1);

  var respInsert boolean;
  var IF_result_CoinBuffer_cbCoinsToStore boolean;
  var IF_result_Control_cInsertCoin boolean;
  var IF_result_TicketStore_tsIsReadyToPrint boolean;
  var IF_result_CoinTray_ctEmptyCoinTray boolean;
  var IF_tmp_target pid;
var IF_sender pid;
var respRequestTicket boolean;
var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_result_TicketMachine_returnCoins boolean;
var IF_result_TicketMachine_printTicket boolean;
var IF_result_TicketTray_ttIsEmpty boolean;
var IF_result_TicketMachine_insertCoin boolean;
var IF_result_Control_cReturnCoins boolean;
var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_Control_cRequestTicket boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var ticketsReadyToPrint boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;
var IF_result_TicketTray_ttEmptyTray boolean;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var respReturnCoins boolean;
var coinsToStoreSuccess boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;
var IF_call_sender pid;

type IF_ConnectivityTableLine_coinBuffer_3 = array [1] of pid;

var IF_Connectivity_coinBuffer IF_ConnectivityTable_coinBuffer_3;

procedure IF_Init_coinBuffer;
  fpar out x IF_ConnectivityTable_coinBuffer_3;
  {#
    x [0] [0] = if_pid_mk(if_CoinBuffer_process, 0);
  #}
endprocedure;

procedure IF_coinBuffer;
fpar in x IF_ConnectivityTable_coinBuffer_3, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

procedure IF_Init_ticketMachine;

fpar out x IF_ConnectivityTable_ticketMachine_3;
#
x [0] [0] = if_pid_mk(if_TicketMachine_process, 0);
#
endprocedure;

procedure IF_ticketMachine;

fpar in x IF_ConnectivityTable_ticketMachine_3, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

procedure IF_Init_ticketStore;

fpar in x IF_ConnectivityTable_coinBuffer_3, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

procedure IF_Init_ticketMachine;

fpar out x IF_ConnectivityTable_ticketMachine_3;
#
x [0] [0] = if_pid_mk(if_TicketMachine_process, 0);
#
endprocedure;

procedure IF_ticketMachine;

fpar in x IF_ConnectivityTable_ticketMachine_3, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

procedure IF_Init_ticketStore;
procedure IF_ticketStore;

fpar in x IF_ConnectivityTable_ticketStore_3, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
{
    return x [sourceProcessId] [targetProcessIndex];
}
endprocedure;

state NoNameTag_10 #start ;
    save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
        IF_call_TicketStore_tsPrintTicket, IF_return_TicketStore_tsIsReadyToPrint,
        IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
        IF_return_TicketMachine_removeCoins, IF_call_TicketMachine_insertCoin, IF_DoNothing,
        IF_return_TicketTray_ttIsEmpty, IF_call_TicketMachine_removeTicket,
        IF_columnCoinStore_csReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
        IF_call_TicketMachine_removeCoins, IF_return_Control_cInsertCoin, IF_return_TicketTray_ttReceiveTicket,
        IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
        IF_return_TicketTray_ttReceiveTicket, IF_call_Control_cReceiveCoinsFromBuffer,
        IF_call_Control_cRequestTicket, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
        IF_call_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
        IF_call_TicketMachine_insertCoin, IF_DoNothing,
        IF_return_TicketTray_ttIsEmpty, IF_call_TicketMachine_removeTicket,
        IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
        IF_return_Control_cRequestTicket, IF_return_TicketTray_ttReceiveTicket,
        IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReceiveCoinsFromBuffer,
        IF_call_Control_cRequestTicket, IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
        IF_call_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
        IF_call_TicketMachine_insertCoin, IF_DoNothing,
        IF_return_TicketTray_ttIsEmpty, IF_call_TicketMachine_removeTicket,
        IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
        IF_return_Control_cRequestTicket, IF_return_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
        IF_call_TicketTray_ttIsReadyToPrint, IF_call_TicketMachine_removeTicket,
        IF_call_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
        IF_return_TicketTray_ttIsReadyToPrint, IF_return_CoinTray_ctEmptyCoinTray,
IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_return_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_return_Control_cInsertCoin, IF_return_TicketTray_ttReceiveTicket,
IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
IF_return_CoinStore_csReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
IF_call_CoinStore_csReceiveCoinsFromBuffer, IF_return_CoinTray_ctReceiveCoins,
IF_return_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
IF_return_TicketMachine_returnCoins, IF_call_TicketMachine_printTicket,
IF_return_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
IF_return_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
IF_call_TicketMachine_removeTicket, IF_call_Control_cRequestTicket,
IF_call_Control_cReturnCoins, IF_return_TicketTray_ttReceiveTicket,
IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
IF_call_TicketMachine_removeTicket, IF_return_Control_cInsertCoin,
IF_return_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
input IF_return_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty, IF_return_CoinTray_ctEmptyCoinTray, IF_call_TicketTray_ttIsEmpty, IF_call_CoinTray_ctEmptyCoinTray, IF_call_Control_cInsertCoin, IF_return_CoinBuffer_cbReceiveCoin;

input IF_return_TicketStore_tsPrintTicket(IF_result_TicketStore_tsPrintTicket);

  task respRequestTicket := IF_result_TicketStore_tsPrintTicket;

  task IF_sender := IF_call_sender;
  task IF_tmp_target := IF_sender;
  output IF_return_Control_cRequestTicket (respRequestTicket) to IF_tmp_target;

  task IF_sender := nil;

  nextstate waitingForRequest;

endstate;

state IF_state_IF_call_34 ;


input IF_return_CoinBuffer_cbReceiveCoin(IF_result_CoinBuffer_cbReceiveCoin);

  task respInsert := IF_result_CoinBuffer_cbReceiveCoin;

  task IF_sender := IF_call_sender;
  task IF_tmp_target := IF_sender;
  output IF_return_Control_cInsertCoin (respInsert) to IF_tmp_target;

  task IF_sender := nil;
nextstate waitingForRequest;
endstate;

state IF_state_IF_call_33;

save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_TicketStore_tsIsReadyToPrint, IF_return_TicketStore_tsIsReadyToPrint,
IF_return_Control_cReturnCoins, IF_call_TicketMachine_removeTicket,
IF_call_TicketMachine_removeCoin, IF_call_TicketMachine_returnCoin,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_call_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_call_TicketTray_ttReceiveTicket, IF_call_TicketTray_ttReceiveTicket,
IF_call_CoinBuffer_cbReceiveCoin, IF_call_Control_cRequestTickets,
IF_call_CoinBuffer_cbReceiveCoin, IF_call_Control_cRequestTicket,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_removeCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_printTicket,
IF_call_TicketMachine_insertCoin, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_call_CoinBuffer_cbReceiveCoin, IF_call_CoinBuffer_cbReceiveCoin;

input IF_result_CoinBuffer_cbCoinsToTray(IF_result_CoinBuffer_cbCoinsToTray);

task respReturnCoins := IF_result_CoinBuffer_cbCoinsToTray;

input IF_result_CoinBuffer_cbCoinsToTray(IF_result_CoinBuffer_cbCoinsToTray);

nextstate waitingForRequest;
endstate;

state IF_state_IF_call_32;

save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_TicketStore_tsIsReadyToPrint, IF_return_TicketStore_tsIsReadyToPrint,
IF_return_Control_cReturnCoins, IF_call_TicketMachine_removeTicket,
IF_call_TicketMachine_removeTicket, IF_call_TicketMachine_removeTicket,
IF_call_TicketMachine_removeCoin, IF_call_TicketMachine_removeCoin,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_printTicket,
IF_call_TicketMachine_insertCoin, IF_call_TicketTray_ttEmptyTray,
IF_call_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_call_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_call_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_call_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
input IF_return_TicketStore_tsIsReadyToPrint(IF_result_TicketStore_tsIsReadyToPrint);

    task ticketsReadyToPrint := IF_result_TicketStore_tsIsReadyToPrint;

nextstate IF_branch_state_0;
endstate;

state waitingForRequest ;

input IF_call_Control_cReturnCoins(IF_sender);
    task IF_call_sender := IF_sender;
    IF_tmp_target := call IF_coinBuffer(IF_Connectivity_coinBuffer, {integer} self, 0);
    output IF_call_CoinBuffer_cbCoinsToTray (self) to IF_tmp_target;
nextstate IF_state_IF_call_33;

input IF_call_Control_cInsertCoin(IF_sender);
    task IF_call_sender := IF_sender;
    IF_tmp_target := call IF_coinBuffer(IF_Connectivity_coinBuffer, {integer} self, 0);
    output IF_call_CoinBuffer_cbReceiveCoin (self) to IF_tmp_target;
nextstate IF_state_IF_call_34;

input IF_call_Control_cRequestTicket(IF_sender);
    task IF_call_sender := IF_sender;
    IF_tmp_target := call IF_ticketStore(IF_Connectivity_ticketStore, {integer} self, 0);
    output IF_call_TicketStore_tsIsReadyToPrint (self) to IF_tmp_target;
    task IF_sender := nil;
nextstate IF_state_IF_call_32;
endstate;

state IF_branch_state_0 unstable ;
    provided ticketsReadyToPrint = true ;
    IF_tmp_target := call IF_coinBuffer(IF_Connectivity_coinBuffer, {integer} self, 0);
    output IF_call_CoinBuffer_cbCoinsToStore (self) to IF_tmp_target;
nextstate IF_state_IF_call_47;
    provided ticketsReadyToPrint = false ;
    task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_Control_cRequestTicket (false) to IF_tmp_target;
task IF_sender := nil;
nextstate waitingForRequest;
endstate;
endprocess;

process TicketTray (1);

var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;
var IF_result_TicketTray_ttEmptyTray boolean;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var IF_result_TicketMachine_returnCoins boolean;
var IF_tmp_target pid;
var IF_result_TicketMachine_printTicket boolean;
var IF_sender pid;
var IF_result_TicketMachine_insertCoin boolean;
var IF_result_Control_cReturnCoins boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;
var IF_result_Control_cRequestTicket boolean;
var IF_result_CoinBuffer_cbCoinsToStore boolean;
var IF_result_TicketStore_tsIsReadyToPrint boolean;
var IF_result_CoinTray_ctEmptyCoinTray boolean;
var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_result_Control_cInsertCoin boolean;
var IF_call_sender pid;
var IF_result_TicketTray_ttIsEmpty boolean;

type IF_ConnectivityTableLine_ticketMachine_4 = array [1] of pid;
type IF_ConnectivityTable_ticketMachine_4 = array [1] of
IF_ConnectivityTableLine_ticketMachine_4;

var IF_Connectivity_ticketMachine IF_ConnectivityTable_ticketMachine_4;

procedure IF_Init_ticketMachine;
  fpar out x IF_ConnectivityTable_ticketMachine_4;
  {
    x [0] [0] = if_pid_mk(if_TicketMachine_process, 0);
  }
endprocedure;

procedure IF_ticketMachine;
  fpar in x IF_ConnectivityTable_ticketMachine_4, in sourceProcessId integer, in
  targetProcessIndex integer;
  returns pid;
  {
    return x [sourceProcessId] [targetProcessIndex];
  }
endprocedure;

type IF_ConnectivityTableLine_ticketStore_4 = array [1] of pid;

type IF_ConnectivityTable_ticketStore_4 = array [1] of
  IF_ConnectivityTableLine_ticketStore_4;

var IF_Connectivity_ticketStore IF_ConnectivityTable_ticketStore_4;

procedure IF_Init_ticketStore;
  fpar out x IF_ConnectivityTable_ticketStore_4;
  {
    x [0] [0] = if_pid_mk(if_TicketStore_process, 0);
  }
endprocedure;

procedure IF_ticketStore;
  fpar in x IF_ConnectivityTable_ticketStore_4, in sourceProcessId integer, in
  targetProcessIndex integer;
returns pid;

{#
    return x [sourceProcessId] [targetProcessIndex];
#}
endprocedure;

state NoNameTag_35 #start ;

save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_TicketStore_tsPrintTicket, IF_return_TicketStore_tsIsReadyToPrint,
IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_DoNothing,
IF_return_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_return_TicketTray_ttReceiveTicket, IF_call_TicketTray_ttReceiveTicket,
IF_return_Control_cReturnCoins, IF_call_Control_cReturnCoins,
IF_call_Control_cInsertCoin, IF_return_TicketTray_ttReceiveTicket,
IF_call_Control_cRequestTicket, IF_return_CoinTray_ctReceiveCoin,
IF_call_CoinTray_ctReceiveCoin, IF_return_CoinTray_ctInsertCoinToTray,
IF_call_CoinBuffer_cbCoinsToTray, IF_return_CoinBuffer_cbCoinsToStore,
IF_call_CoinBuffer_cbCoinsToStore, IF_return_TicketMachine_removeTicket,
IF_return_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
IF_return_TicketMachine_printTicket, IF_return_TicketMachine_insertCoin,
IF_call_Control_cReturnCoins, IF_return_Control_cRequestTicket,
IF_call_CoinStore_csReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
IF_call_CoinStore_csReceiveCoinsFromBuffer, IF_return_CoinTray_ctReceiveCoin,
IF_call_CoinBuffer_cbCoinsToTray, IF_return_CoinBuffer_cbCoinsToStore,
IF_call_CoinBuffer_cbCoinsToStore, IF_return_TicketMachine_removeTicket,
IF_return_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
IF_return_TicketMachine_printTicket, IF_return_TicketMachine_insertCoin,
IF_return_Control_cInsertCoin, IF_return_TicketMachine_removeTicket,
IF_return_TicketTray_ttIsEmpty, IF_call_TicketTray_ttIsEmpty,
IF_return_Control_cReceiveCoinsFromBuffer, IF_return_Control_cReceiveCoins,
IF_call_CoinBuffer_cbReceiveCoin;

call IF_Init_ticketStore(IF_Connectivity_ticketStore);

call IF_Init_ticketMachine(IF_Connectivity_ticketMachine);

task IF_sender := nil;

nextstate ready;

endstate;

state ready ;

input IF_call_TicketTray_ttIsEmpty(IF_sender);

    task IF_call_sender := IF_sender;

    task IF_sender := IF_call_sender;

    task IF_tmp_target := IF_sender;

    output IF_return_TicketTray_ttIsEmpty (true) to IF_tmp_target;

    task IF_sender := nil;

nextstate ready;

input IF_call_TicketTray_ttEmptyTray(IF_sender);

    task IF_call_sender := IF_sender;

    task IF_sender := IF_call_sender;

    task IF_tmp_target := IF_sender;
output IF_return_TicketTray_ttEmptyTray (false) to IF_tmp_target;

task IF_sender := nil;

nextstate ready;

input IF_call_TicketTray_ttReceiveTicket(IF_sender);

task IF_call_sender := IF_sender;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketTray_ttReceiveTicket (true) to IF_tmp_target;

task IF_sender := nil;

nextstate full;
endstate;

state full ;

input IF_call_TicketTray_ttIsEmpty(IF_sender);

task IF_call_sender := IF_sender;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketTray_ttIsEmpty (false) to IF_tmp_target;

task IF_sender := nil;

nextstate full;

input IF_call_TicketTray_ttReceiveTicket(IF_sender);

task IF_call_sender := IF_sender;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketTray_ttReceiveTicket (false) to IF_tmp_target;

task IF_sender := nil;

nextstate full;

input IF_call_TicketTray_ttEmptyTray(IF_sender);

task IF_call_sender := IF_sender;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketTray_ttEmptyTray (true) to IF_tmp_target;

task IF_sender := nil;

nextstate ready;
endstate;
endprocess;

process CoinTray (1);

var IF_result_Control_cInsertCoin boolean;
var coinsInTray CoinsInTray public;
var IF_tmp_target pid;
var IF_sender pid;
var IF_result_CoinBuffer_cbCoinsToStore boolean;
var IF_result_TicketStore_tsIsReadyToPrint boolean;
var IF_result_CoinTray_ctEmptyCoinTray boolean;
var coins integer;
var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_result_TicketMachine_returnCoins boolean;
var IF_result_TicketMachine_printTicket boolean;
var IF_result_TicketMachine_insertCoin boolean;
var IF_result_Control_cReturnCoins boolean;
var IF_result_Control_cRequestTicket boolean;
var IF_result_TicketTray_ttIsEmpty boolean;
var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;
var IF_result_TicketTray_ttEmptyTray boolean;
var IF_call_sender pid;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;

type IF_ConnectivityTableLine_coinBuffer_5 = array [1] of pid;

var IF_Connectivity_coinBuffer IF_ConnectivityTable_coinBuffer_5;
procedure IF_Init_coinBuffer;
  fpar out x IF_ConnectivityTable_coinBuffer_5;
  {#
    x [0] [0] = if_pid_mk(if_CoinBuffer_process, 0);
  #}
endprocedure;

procedure IF_coinBuffer;
  fpar in x IF_ConnectivityTable_coinBuffer_5, in sourceProcessId integer, in
targetProcessIndex integer;
  returns pid;
  {#
    return x [sourceProcessId] [targetProcessIndex];
  #}
endprocedure;

type IF_ConnectivityTableLine_ticketMachine_5 = array [1] of pid;

  type IF_ConnectivityTable_ticketMachine_5 = array [1] of
    IF_ConnectivityTableLine_ticketMachine_5;

  var IF_Connectivity_ticketMachine IF_ConnectivityTable_ticketMachine_5;

procedure IF_Init_ticketMachine;
  fpar out x IF_ConnectivityTable_ticketMachine_5;
  {#
    x [0] [0] = if_pid_mk(if_TicketMachine_process, 0);
  #}
endprocedure;

procedure IF_ticketMachine;
  fpar in x IF_ConnectivityTable_ticketMachine_5, in sourceProcessId integer, in
  targetProcessIndex integer;
  returns pid;
  {#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

state NoNameTag_21 #start ;
    save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
    IF_call_TicketStore_tIsReadyToPrint, IF_return_Control_cRequestTicket,
    IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
    IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin, IF_DoNothing,
    IF_return_TicketTray_tIsEmptyTray, IF_call_TicketTray_tIsEmptyTray,
    IF_return_Control_cInsertCoin, IF_return_TicketTray_tReceiveTicket,
    IF_call_TicketTray_tReceiveTicket, IF_call_Control_cReturnCoins,
    IF_return_CoinStore_cReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
    IF_call_CoinStore_cReceiveCoinsFromBuffer, IF_return_CoinTray_tReceiveCoins,
    IF_call_CoinTray_tReceiveCoins, IF_return_CoinBuffer_cbCoinsToTray,
    IF_call_Control_cInsertCoin, IF_return_CoinBuffer_cbCoinsToStore,
    IF_call_CoinBuffer_cbCoinsToStore, IF_return_TicketMachine_removeTicket,
    IF_return_TicketMachine_removeCoins, IF_return_TicketMachine_returnCoins,
    IF_call_CoinTray_tIsEmpty, IF_return_CoinTray_tIsEmpty,
    IF_call_CoinTray_tIsEmpty, IF_call_Control_cInsertCoin,
    IF_call_Control_cInsertCoin, IF_return_CoinBuffer_cbReceiveCoin;
    call IF_Init_ticketMachine(IF_Connectivity_ticketMachine);
    call IF_Init_coinBuffer(IF_Connectivity_coinBuffer);
    task IF_sender := nil;
    task coinsInTray := 0;
    nextstate coinTrayNotFull;
endstate;

state IF_branch_state_7 #unstable ;
    provided coinsInTray = COINTRAY_CAP ;
    task IF_sender := IF_call_sender;
    task IF_tmp_target := IF_sender;
    output IF_return_CoinTray_ctReceiveCoins (true) to IF_tmp_target;
    task IF_sender := nil;
    nextstate coinTrayFull;
    provided coinsInTray > COINTRAY_CAP ;
    task coinsInTray := coinsInTray - coins;
    task IF_sender := IF_call_sender;
    task IF_tmp_target := IF_sender;
    output IF_return_CoinTray_ctReceiveCoins (false) to IF_tmp_target;
task IF_sender := nil;
nextstate coinTrayNotFull;

provided coinsInTray < COINTRAY_CAP ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinTray_ctReceiveCoins (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinTrayNotFull;
endstate;

state coinTrayNotFull ;

input IF_call_CoinTray_ctReceiveCoins(coins, IF_sender);
task IF_call_sender := IF_sender;
task coinsInTray := coinsInTray + coins;
task IF_sender := nil;
nextstate IF_branch_state_7;

input IF_call_CoinTray_ctEmptyCoinTray(IF_sender);
task IF_call_sender := IF_sender;
task coinsInTray := 0;

nextstate coinTrayFull;

endstate;

state coinTrayFull ;

input IF_call_CoinTray_ctEmptyCoinTray(IF_sender);
task IF_call_sender := IF_sender;
task coinsInTray := 0;

nextstate coinTrayNotFull;

endstate;
task IF_sender := nil;
nextstate coinTrayNotFull;
input IF_call_CoinTray_ctReceiveCoins(coins, IF_sender);
task IF_call_sender := IF_sender;
task IF_sender := nil;
nextstate IF_branch_state_8;
endstate;
state IF_branch_state_8 #unstable ;
provided coins = 0 ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinTray_ctReceiveCoins (true) to IF_tmp_target;
task IF_sender := nil;
nextstate coinTrayFull;
provided coins > 0 ;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_CoinTray_ctReceiveCoins (false) to IF_tmp_target;
task IF_sender := nil;
nextstate coinTrayFull;
endstate;
endprocess;

process TicketStore (1);

var IF_result_CoinStore_csReceiveCoinsFromBuffer boolean;
var IF_tmp_target pid;
var IF_sender pid;
var IF_result_TicketMachine_returnCoins boolean;
var IF_result_TicketMachine_printTicket boolean;
var IF_result_TicketMachine_insertCoin boolean;
var IF_result_Control_cReturnCoins boolean;
var IF_result_TicketTray_ttIsEmpty boolean;
var IF_result_Control_cRequestTicket boolean;
var IF_result_CoinBuffer_cbReceiveCoin boolean;
var IF_result_TicketStore_tsPrintTicket boolean;
var IF_result_CoinTray_ctReceiveCoins boolean;
var IF_result_TicketTray_ttEmptyTray boolean;
var IF_result_CoinBuffer_cbCoinsToTray boolean;
var trayHasAccepted boolean;
var trayCanAccept boolean;
var IF_result_TicketTray_ttReceiveTicket boolean;
var ticketsInStore TicketsInStore public;
var IF_call_sender pid;
var IF_result_Control_cInsertCoin boolean;
var IF_result_CoinBuffer_cbCoinsToStore boolean;
var IF_result_TicketStore_tsIsReadyToPrint boolean;
var IF_result_CoinTray_ctEmptyCoinTray boolean;

type IF_ConnectivityTableLine_ticketTray_6 = array [1] of pid;

procedure IF_Init_ticketTray;
fpar out x IF_ConnectivityTable_ticketTray_6;
{#
  x [0] [0] = if_pid_mk(if_TicketTray_process, 0);
#}
endprocedure;

procedure IF_ticketTray;
fpar in x IF_ConnectivityTable_ticketTray_6, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;
{#
return x [sourceProcessId] [targetProcessIndex];
#
endprocedure;

procedure IF_Init_control;

fpar out x IF_ConnectivityTable_control_6;

{#
x [0] [0] = if_pid_mk(if_Control_process, 0);
#
}endprocedure;

procedure IF_control;

fpar in x IF_ConnectivityTable_control_6, in sourceProcessId integer, in targetProcessIndex integer;
returns pid;

{#
return x [sourceProcessId] [targetProcessIndex];
#
}endprocedure;

state NoNameTag_29 #start ;

IF_call_Control_cInsertCoin, IF_return_CoinBuffer_cbReceiveCoin;

call IF_Init_control(IF_Connectivity_control);

call IF_Init_ticketTray(IF_Connectivity_ticketTray);

task IF_sender := nil;

task ticketsInStore := TICKETSTORE_CAP;

nextstate notEmpty;

endstate;

state notEmpty ;

input IF_call_TicketStore_tsIsReadyToPrint(IF_sender);

task IF_call_sender := IF_sender;

IF_tmp_target := call IF_ticketTray(IF_Connectivity_ticketTray, {integer}
self, 0);

output IF_call_TicketTray_ttIsEmpty (self) to IF_tmp_target;

task IF_sender := nil;

nextstate IF_state_IF_call_41;

endstate;

state IF_branch_state_11 #unstable ;

provided ticketsInStore > 0 ;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketStore_tsPrintTicket (true) to IF_tmp_target;

task IF_sender := nil;

nextstate notEmpty;

provided ticketsInStore <= 0 ;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketStore_tsPrintTicket (true) to IF_tmp_target;

task IF_sender := nil;

nextstate emptyTicketStore;

endstate;

state emptyTicketStore ;

input IF_call_TicketStore_tsPrintTicket(IF_sender);

task IF_call_sender := IF_sender;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_TicketStore_tsPrintTicket (false) to IF_tmp_target;
task IF_sender := nil;
nextstate emptyTicketStore;

input IF_call_TicketStore_tsIsReadyToPrint (IF_sender);
task IF_call_sender := IF_sender;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_TicketStore_tsIsReadyToPrint (false) to IF_tmp_target;
task IF_sender := nil;
nextstate emptyTicketStore;

endstate;

state IF_state_IF_call_41 ;

save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore_tsPrintTicket,
IF_call_TicketStore_tsPrintTicket, IF_return_TicketStore_tsIsReadyToPrint,
IF_call_TicketStore_tsIsReadyToPrint, IF_return_Control_cReturnCoins,
IF_call_TicketMachine_removeTicket, IF_return_Control_cRequestTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_return_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_return_Control_cInsertCoin, IF_return_TicketTray_ttReceiveTicket,
IF_call_TicketTray_ttReceiveTicket, IF_call_Control_cReturnCoins,
IF_return_CoinBuffer_cbReceiveCoin, IF_call_Control_cRequestTicket,
IF_call_CoinStore_csReceiveCoinsFromBuffer, IF_call_Control_cRequestTicket,
IF_call_CoinTray_ctReceiveCoinsFromBuffer, IF_return_CoinBuffer_cbReceiveCoin,
IF_call_CoinTray_ctReceiveCoins, IF_return_CoinBuffer_cbCoinsToTray,
IF_call_CoinBuffer_cbCoinsToTray, IF_return_CoinBuffer_cbCoinsToStore,
IF_call_CoinBuffer_cbCoinsToStore, IF_return_TicketMachine_removeTicket,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_return_TicketTray_ttIsEmpty, IF_call_TicketMachine_removeTicket,
IF_return_TicketMachine_printTicket, IF_return_TicketMachine_insertCoin,
IF_call_TicketTray_ttEmptyCoinTray, IF_call_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttIsEmpty, IF_call_Control_cInsertCoin,
IF_return_CoinBuffer_cbReceiveCoin;

input IF_return_TicketTray_ttIsEmpty (IF_result_TicketTray_ttIsEmpty);
task trayCanAccept := IF_result_TicketTray_ttIsEmpty;

nextstate IF_branch_state_10;
endstate;

state readyToPrintTicket ;

input IF_call_TicketStore_tsIsReadyToPrint (IF_sender);
task IF_call_sender := IF_sender;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;

198
output IF_return_TicketStore_tsIsReadyToPrint(true) to IF_tmp_target;

nextstate readyToPrintTicket;

input IF_call_TicketStore_tsPrintTicket(IF_sender);
task IF_call_sender := IF_sender;
task ticketsInStore := ticketsInStore - 1;

IF_tmp_target := call IF_ticketTray(IF_Connectivity_ticketTray, {integer}
self, 0);
output IF_call_TicketTray_ttReceiveTicket(self) to IF_tmp_target;
task IF_sender := nil;
nextstate IF_state_IF_call_38;
endstate;

state IF_state_IF_call_38;
save IF_call_CoinBuffer_cbReceiveCoin, IF_return_TicketStore tsPrintTicket,
IF_call_TicketStore_tsPrintTicket, IF_return_TicketStore_tsIsReadyToPrint,
IF_call_TicketMachine_removeTicket, IF_return_Control_cReturnCoins,
IF_call_TicketMachine_removeCoins, IF_call_TicketMachine_returnCoins,
IF_call_TicketMachine_printTicket, IF_call_TicketMachine_insertCoin,
IF_return_TicketTray_ttEmptyTray, IF_call_TicketTray_ttEmptyTray,
IF_return_Control_cInsertCoin, IF_call_TicketTray_ttReceiveTicket,
IF_call_Control_cReturnCoins, IF_return_CoinStore_csReceiveCoinsFromBuffer,
IF_call_Control_cRequestTicket, IF_call_CoinStore_csReceiveCoinsFromBuffer,
IF_return_CoinTray_ctReceiveCoins, IF_call_CoinTray_ctReceiveCoins,
IF_return_CoinBuffer_cbCoinsToTray, IF_call_CoinBuffer_cbCoinsToTray,
IF_return_CoinBuffer_cbCoinsToStore, IF_call_CoinBuffer_cbCoinsToStore,
IF_return_TicketMachine_removeTicket, IF_return_TicketMachine_removeCoins,
IF_return_TicketMachine_returnCoins, IF_return_TicketMachine_printTicket,
IF_return_TicketMachine_insertCoin, IF_return_TicketTray_ttIsEmpty,
IF_call_TicketTray_ttEmptyCoinTray, IF_call_Control_cInsertCoin,
IF_call_Control_cRequestTicket, IF_call_CoinStore csReceiveCoinsFromBuffer,
IF_return_CoinTray ctEmptyCoinTray, IF_call_CoinTray ctEmptyCoinTray,
IF_call_Control cInsertCoin,
IF_return_CoinBuffer cbReceiveCoin;

input
IF_return_TicketTray_ttReceiveTicket(IF_result_TicketTray_ttReceiveTicket);

task trayHasAccepted := IF_result_TicketTray_ttReceiveTicket;

nextstate IF_branch_state_11;
endstate;

state IF_branch_state_10 unstable;

provided trayCanAccept = false;
task IF_sender := IF_call_sender;
task IF_tmp_target := IF_sender;
output IF_return_TicketStore_tsIsReadyToPrint(false) to IF_tmp_target;
task IF_sender := nil;

nextstate notEmpty;

provided trayCanAccept = true ;

task IF_sender := IF_call_sender;

task IF_tmp_target := IF_sender;

output IF_return_TicketStore_tsIsReadyToPrint (true) to IF_tmp_target;

task IF_sender := nil;

nextstate readyToPrintTicket;

endstate;

endprocess;

endsystem;
Appendix G - MOFScript generated IF 2.0 code

```plaintext
## Compiling 'NewTransformation.m2t' (Using repository path:
reference:file:plugins/org.eclipse.mofscript.editor_1.3.3/repository/metamodels)
## Parsing 'NewTransformation.m2t'
## Finished Compiling.
## No errors
## Open File: C:\Users\trogenhofer\workspace\Getting started with UML2\TicketMachine.uml for input
## model:uml
## Executing MOFScript Specification
## Starting transformation: "SdlUmlToF20Transformation"

system TicketMachine
signalroute0 from TicketMachine to TicketTray with
signal EmptyTicketTray(boolean);;
signalroute0 from TicketTray to TicketMachine with
signal EmptyTicketTray(boolean);;
const TicketPrice = 2;
const CoinStoreCAP = 6;
const CoinBufferCAP = 3;
const CoinTrayCAP = 3;
signal ReceiveCoinsFromBuffer(boolean);
signal ReceiveCoin(coins);
signal CoinsToTray
signal CoinsToStore
signal InsertCoin
signal ReturnCoin
signal RequestTicket(boolean);
signal EmptyCoinTray
signal IsReadyToPrintTicket(boolean);
signal PrintTicket(boolean);
signal TicketTrayIsEmpty(boolean);
signal ReceiveTicket(boolean);
process TicketMachine
found region
state start #start;
nextstate initialized;
state initialized;
input InsertCoin
nextstate initialized;
input EmptyCoinTray
nextstate initialized;
input RequestTicket(boolean);
nextstate initialized;
input ReturnCoin
nextstate initialized;
input EmptyTicketTray
nextstate initialized;
endprocess;
process Control
found region
state start #start;
nextstate WaitingForRequest;
state choiceA #unstable;
provided [ticketsReadyToPrint, =, false]
nextstate WaitingForRequest;
provided [ticketsReadyToPrint]
nextstate choiceB;
```

201
state choiceB #unstable;
provided [coinsToStoreSuccess, =, false]
nextstate WaitingForRequest;
provided [coinsToStoreSuccess, =, true]
nextstate WaitingForRequest;
state WaitingForRequest;
input InsertCoin
nextstate WaitingForRequest;
input ReturnCoins
nextstate WaitingForRequest;
nextstate choiceA;
endprocess;
process CoinBuffer
found region
state start #start;
nextstate coinBufferNotFull;
state choiceA #unstable;
provided [coinsInBuffer, <, CoinBufferCAP]
nextstate coinBufferNotFull;
provided [coinsInBuffer, >=, CoinBufferCAP]
nextstate coinBufferFull;
state choiceB #unstable;
provided [trayHasAccepted, =, false]
nextstate coinBufferFull;
provided [trayHasAccepted, =, true]
nextstate coinBufferNotFull;
state choiceC #unstable;
provided [trayHasAccepted, =, false]
nextstate coinBufferNotFull;
provided [trayHasAccepted, =, true]
nextstate coinBufferNotFull;
state choiceD #unstable;
provided [coinsInBuffer, <, TicketPrice]
nextstate coinBufferNotFull;
provided [coinsInBuffer, >=, TicketPrice]
nextstate coinBufferNotFull;
state choiceE #unstable;
provided [storeHasAccepted, =, false]
nextstate coinBufferFull;
provided [storeHasAccepted, =, true]
nextstate coinBufferNotFull;
state coinBufferNotFull;
input ReceiveCoin(coins);
nextstate choiceA;
input CoinsToTray
nextstate choiceC;
input CoinsToStore
nextstate choiceD;
state coinBufferFull;
input ReceiveCoin(coins);
nextstate coinBufferFull;
input CoinsToTray
nextstate choiceB;
input CoinsToStore
nextstate choiceE;
endprocess;
process CoinTray
found region
state start #start;
nextstate coinTrayNotFull;
state choiceA #unstable;
provided [coinsInTray, >, CoinTrayCAP]
nextstate coinTrayNotFull;
provided [coinsInTray, <, CoinTrayCAP]
nextstate coinTrayNotFull;
provided [coinsInTray, =, CoinTrayCAP]
nextstate coinTrayFull;
state ChoiceB #unstable;
provided [coins, >, 0]
nextstate coinTrayFull;
provided [coins, =, 0]
nextstate coinTrayFull;
state coinTrayNotFull;
input ReceiveCoin(coins);
nextstate choiceA;
input EmptyCoinTray
nextstate coinTrayNotFull;
state coinTrayFull;
input ReceiveCoin(coins);
nextstate ChoiceB;
input EmptyCoinTray
nextstate coinTrayNotFull;
endprocess;
process CoinStore
found region
state start #start;
nextstate coinStoreWaiting;
state choiceA #unstable;
provided [coinsInStore, >=, CoinStoreCAP]
nextstate coinStoreFull;
provided [coinsInStore, <, CoinStoreCAP]
nextstate coinStoreWaiting;
state coinStoreWaiting;
nextstate choiceA;
state coinStoreFull;
nextstate coinStoreFull;
endprocess;
process TicketStore
found region
state start #start;
nextstate notEmpty;
state choiceA #unstable;
provided [trayCanAccept, =, false]
nextstate notEmpty;
provided [trayCanAccept, =, true]
nextstate readyToPrintTicket;
state choiceB #unstable;
provided [ticketsInStore, <=, 0]
nextstate emptyTicketStore;
provided [ticketsInStore, >, 0]
nextstate notEmpty;
state notEmpty;
input IsReadyToPrintTicket(boolean);
nextstate choiceA;
state readyToPrintTicket;
input IsReadyToPrintTicket(boolean);
nextstate readyToPrintTicket;
input PrintTicket(boolean);
nextstate choiceB;
state emptyTicketStore;
input PrintTicket(boolean);
nextstate emptyTicketStore;
input IsReadyToPrintTicket(boolean);
nextstate emptyTicketStore;
endprocess;
process TicketTray
found region
state start #start;
nextstate ready;
state ready;
input ReceiveTicket(boolean);
nexstate full;
input TicketTrayIsEmpty(boolean);
nexstate ready;
input EmptyTicketTray(boolean);
output EmptyTicketTray([false]);
nexstate ready;
state full;
input EmptyTicketTray(boolean);
nexstate ready;
input TicketTrayIsEmpty(boolean);
nexstate full;
input ReceiveTicket(boolean);
nexstate full;
endprocess;
endsystem;
## Finished transformation: "SdlUmlToF20Transformation"
Appendix H - MOFScript transformation template

/**
* MOFScript SDL-UML to IF 2.0 transformation template
* Author: Thomas Rogenhofer
* Date: 28.12.2010
*/

texttransformation SdlUmlToIF20Transformation (in uml:"http://www.eclipse.org/uml2/2.1.0/UML") {

/**
* Main (entry point)
*/

var ownedMemberList:List

uml.Model::main(){
    self.ownedMember->forEach(p:uml.Package){
        ownedMemberList = p.ownedMember
    }

    self.ownedMember->forEach(p:uml.Package){
        p.mapPackage()
    }
}

/**
* SDL-UML ticket machine model to IF 2.0 system transformation
*/

uml.Package::mapPackage(){
    self.ownedMember->forEach(c:uml.Class) c.mapSystemName()
    self.ownedMember->forEach(i:uml.InformationFlow) i.mapConnector()
    self.ownedMember->forEach(c:uml.Class) c.mapConstant()
    self.ownedMember->forEach(s:uml.Signal) s.mapSignalSystemDef()
    self.ownedMember->forEach(c:uml.Class) c.mapActiveClass()
    '\n endsystem;'
}

/**
* Transformation hypothesis 1
* Transformation rule 1
* Description: Transformation of the outermost <<ActiveClass>>
to an IF 2.0 system specification
*/

uml.Class::mapSystemName(){
    if (self.hasStereotype("activeClass")){
        '\n system ' + self.name
    }
/**
 * Transformation hypothesis 2
 * Transformation rule 2
 * Description: Transformation of a SDL-UML <<ActiveClass>> to
 * an IF 2.0 process specification
 */

uml.Class::mapActiveClass(){
    if (self.hasStereotype("activeClass")){
        "\n process ' self.name '(1)''
        self.mapStateMachine()
        "\n endprocess;''
        //"flatten" the composite structure
        self.ownedMember->forEach(c:uml.Class){
            if (c.hasStereotype("activeClass")){
                "\n process ' c.name '(1)''
                c.mapStateMachine()
                "\n endprocess;''
            }
        }
    }
}

/**
 * Transformation hypothesis 3
 * Transformation rule 3
 * Description: a <<Connector>> with an associated
 * InformationFlow that has an
 * association to <<Signal>> can be transformed a Signalroute
 * for the purpose of
 * automatic test case generation.
 */

uml.InformationFlow::mapConnector(){
    var counter:Integer = 0
    "\n signalroute sig' + counter++ ' from ''
    //Identify <<activeClass>> that owns the port from which
    signal is sent
    self.informationSource->forEach(pol:uml.Port){
        ownedMemberList->forEach(cl:uml.Class){
            if (cl.hasStereotype("activeClass")){
                cl.ownedAttribute->forEach(pol2:uml.Port){
                    if (pol = pol2){
                        cl.name
                    }
                }
            }
        }
        cl.ownedMember->forEach(c:uml.Class){
            if (c.hasStereotype("activeClass")){
                c.ownedAttribute->forEach(pol3:uml.Port){
                    if (pol = pol3){
                        c2.name
                    }
                }
            }
        }
    }
}
}
Identify <<activeClass>> that owns the port on which signal is received

```java
self.informationTarget->forEach(pol:uml.Port)
  ownedMemberList->forEach(c1:uml.Class)
    if (c1.hasStereotype("activeClass"))
      if (pol = po2)
        c1.name
  }

c1.ownedMember->forEach(c2:uml.Class)
  if (c2.hasStereotype("activeClass"))
    c2.ownedAttribute->forEach(po3:uml.Port)
      if (po1 = po3)
        c2.name
    }
  }
}

//identify signals that are being sent over signalroute
self.conveyed->forEach(s1:uml.Signal)
  ownedMemberList->forEach(s2:uml.Signal)
    if (s1 = s2)
      s2.mapSignalSystemDef()
}
'

**
* Transformation hypothesis 3
* Transformation rule 4
* Description: Function that supports the transformation of Sa DL-UML <<Connectors>> to an IF 2.0
* signalroute specifications. This function identifies the <<activeClass>> for a specific port
*/

uml.Class::mapPort(pol:uml.Port)
  if (self.hasStereotype("activeClass"))
    self.ownedAttribute->forEach(po2:uml.Port)
      if (po1 = po2)
        self.name
    }
  }

self.ownedMember->forEach(c:uml.Class)
  if (c.hasStereotype("activeClass"))
    c.ownedAttribute->forEach(po3:uml.Port)
      if (po1 = po3)
        c.name
```
/**
 * Transformation hypothesis 4
 * Transformation rule 5
 * Description: Transformation of a SDL-UML stereotype
 * <<StateMachine>> to a number of IF 2.0 process states
 */

uml.Class::mapStateMachine(){
    self.ownedBehavior->forEach(s:uml.StateMachine){
        s.ownedMember->forEach(r:uml.Region){
            r.mapPseudoStateInitial()
            r.mapPseudoStateChoice()
            r.mapState()
        }
    }
}

/**
 * Transformation hypothesis 4
 * Transformation rule 6
 * Description: Transformation of a SDL-UML stereotype
 * <<PseudoState>> of kind "initial"
 * to an IF 2.0 state of type "#start"
 */

uml.Region::mapPseudoStateInitial(){
    self.subvertex->forEach(psi:uml.Pseudostate){
        if (psi.kind = "initial"){
            '\n state ' psi.name ' #start;'
            self.transition->forEach(t:uml.Transition){
                t.mapTransition(psi)
            }
        }
    }
}

/**
 * Transformation hypothesis 4
 * Transformation rule 7
 * Description: Transformation of a SDL-UML stereotype
 * <<PseudoState>> of kind "choice"
 * to an IF 2.0 state of type "#unstable"
 */

uml.Region::mapPseudoStateChoice(){
    self.subvertex->forEach(psc:uml.Pseudostate){
        if (psc.kind = "choice"){
            '\n state ' psc.name ' #unstable;'
            self.transition->forEach(t:uml.Transition){
                t.mapTransition(psc)
            }
        }
    }
}
Transformation hypothesis 4
Transformation rule 8
Description: Transformation of a SDL-UML stereotype <<State>> to an IF 2.0 state specification

```java
uml.Region::mapState()
    self.subvertex->forEach(st:uml.State)
        '\n state ' st.name ';'
        self.transition->forEach(t:uml.Transition)
            t.mapTransition(st)
}
```

Transformation hypothesis 4
Transformation rule 9
Description: Transformation of a SDL-UML stereotype <<Transition>> to an IF 2.0 transition specifications

```java
uml.Transition::mapTransition(v:uml.Pseudostate)
    var var1:String
    //Transform SDL-UML signal event that triggers the transition
    //to IF 2.0 signal event
    if (self.source = v){
        self.trigger->forEach(tr:uml.Trigger){
            ownedMemberList->forEach(e:uml.ReceiveSignalEvent){
                if (tr.event = e){
                    ownedMemberList->forEach(s:uml.Signal){
                        if (e.signal = s){
                            s.mapSignalProcessRec()
                        }
                    }
                }
            }
        }
    }
    //Transform SDL-UML transition guard to IF 2.0 transition
guard
    if(self.source = v){
        self.ownedRule->forEach(co:uml.Constraint){
            co.ownedElement->forEach(oe:uml.OpaqueExpression)
                '\n provided ' oe.body
        }
    }
    //Transform a SDL-UML activity to IF 2.0 process tasks
if(self.source = v){
    self.ownedElement->forEach(ac:uml.Activity){
        ac.group->forEach(sq:uml.SequenceNode){
            sq.executableNode->forEach(sa:uml.SendSignalAction){
                sa.argument->forEach(vp:uml.ValuePin){
                    ownedMemberList->forEach(so:uml.Signal){
```
if (so = sa.signal) {
    \n    so.mapSignalProcessSend(vp.value.body)
}

sa.ownedElement->forEach(ai:uml.ActionInputPin) {
    
}

// Transform the SDL-UML target state to an IF 2.0 target state
self.source = v) {
    \n    nextstate 'self.target.name';
}

/**
 * Transformation hypothesis 5
 * Transformation rule 10
 * Description: a <<Signal>> can be transformed to Signal
 */

// IF 2.0 system signal definition
uml.Signal::mapSignalSystemDef() {
    \n    signal 'self.name self.ownedAttribute->forEach(prs:uml.Property) {
        (' prs.name ');
    }
}

// IF 2.0 process signal reception definition
uml.Signal::mapSignalProcessRec() {
    \n    input 'self.name self.ownedAttribute->forEach(prs:uml.Property) {
        (' prs.name ');
    }
}

// IF 2.0 process signal send definition
uml.Signal::mapSignalProcessSend(v: String) {
    \n    output 'self.name self.ownedAttribute->forEach(prc:uml.Property) {
        (' v ');
    }
}

/**
 * Transformation hypothesis 6
 * Transformation rule 11
 * Description: <<PrimitiveType>> PrimitiveType with name = Integer can be transformed to Integer,
 * <<PrimitiveType>> PrimitiveType with name = Boolean can be transformed to Boolean.
 */

uml.Class::mapConstant() {
    self.ownedAttribute->forEach(prc:uml.Property) {
if(prc.isReadOnly) '\n const ' prc.name ' = '
prc.ownedElement->forEach(i:uml.LiteralInteger){
  i.value ';
}
}
Appendix I - Abstract test case for verification method

<?xml version="1.0" encoding="UTF-8"?>
<testSuite>
  <abstractTestSuite generator="TGV">
    <model>
      <class name="global">
        <constants>
          <constant name="COINSTORE_CAP" type="int">
            <value>6</value>
          </constant>
          <constant name="COINTRAY_CAP" type="int">
            <value>3</value>
          </constant>
          <constant name="TICKETTRAY_CAP" type="int">
            <value>1</value>
          </constant>
          <constant name="TICKETSTORE_CAP" type="int">
            <value>3</value>
          </constant>
          <constant name="COINBUFFER_CAP" type="int">
            <value>3</value>
          </constant>
          <constant name="TICKET_PRICE" type="int">
            <value>2</value>
          </constant>
          <constant name="INIT_TICKETS_IN_STORE" type="int">
            <value>3</value>
          </constant>
        </constants>
      </class>
    </model>
  </abstractTestSuite>
</testSuite>
<types>
  <type name="TicketsInStore">
    <range type="int">
      <interval from="0" to="4"/>
    </range>
  </type>
  <type name="CoinsInStore">
    <range type="int">
      <interval from="0" to="7"/>
    </range>
  </type>
  <type name="CoinsInTray">
    <range type="int">
      <interval from="0" to="4"/>
    </range>
  </type>
</types>
</class>
<class name="TicketMachine">
  <members>
    <member signature="insertCoin():bool"/>
    <member signature="returnCoins():bool"/>
    <member signature="removeCoins()"/>
    <member signature="printTicket():bool"/>
    <member signature="removeTicket()"/>
  </members>
</class>
<object name="theTicketMachine" class="TicketMachine" environment="false"/>
<testCase id="tc1">
  <step id="T0" nextPass="T1">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
      </interaction>
    </step>
  <step id="T1" nextPass="T2">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
      <value>true</value>
    </interaction>
  </step>
  <step id="T2" nextPass="T3">
    <interaction object="theTicketMachine" signature="returnCoins():bool" type="call">
      </interaction>
    </step>
  <step id="T3" nextPass="T4">
    <interaction object="theTicketMachine" signature="returnCoins():bool" type="return">
      <value>true</value>
    </interaction>
  </step>
  <step id="T4" nextPass="T5">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
      </interaction>
    </step>
  <step id="T5" nextPass="T6">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
      <value>true</value>
    </interaction>
  </step>
</testCase>
<step id="T6" nextPass="T7">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
  </interaction>
</step>

<step id="T7" nextPass="T8">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
    <value>true</value>
  </interaction>
</step>

<step id="T8" nextPass="T9">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
  </interaction>
</step>

<step id="T9" nextPass="T10">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
    <value>true</value>
  </interaction>
</step>

<step id="T10" nextPass="T11">
  <interaction object="theTicketMachine" signature="removeCoins()" type="call">
  </interaction>
</step>

<step id="T11" nextPass="T12">
  <interaction object="theTicketMachine" signature="removeCoins()" type="return">
  </interaction>
</step>
<step id="T12" nextPass="T13">
    <interaction object="theTicketMachine" signature="returnCoins():bool" type="call">
    
    </interaction>
</step>
<step id="T13" nextPass="T14">
    <interaction object="theTicketMachine" signature="returnCoins():bool" type="return">
        <value>true</value>
    </interaction>
</step>
<step id="T14" nextPass="T15">
    <interaction object="theTicketMachine" signature="printTicket():bool" type="call">
    </interaction>
</step>
<step id="T15" nextPass="T16">
    <interaction object="theTicketMachine" signature="printTicket():bool" type="return">
        <value>false</value>
    </interaction>
</step>
<step id="T16" nextPass="T17">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
    </interaction>
</step>
<step id="T17" nextPass="T18">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
        <value>true</value>
    </interaction>
</step>
<step id="T18" nextPass="T19">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
    </interaction>
</step>

<step id="T19" nextPass="T20">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
        <value>true</value>
    </interaction>
</step>

<step id="T20" nextPass="T21">
    <interaction object="theTicketMachine" signature="returnCoins():bool" type="call">
    </interaction>
</step>

<step id="T21" nextPass="T22">
    <interaction object="theTicketMachine" signature="returnCoins():bool" type="return">
        <value>false</value>
    </interaction>
</step>

<step id="T22" nextPass="T23">
    <interaction object="theTicketMachine" signature="removeCoins()" type="call">
    </interaction>
</step>

<step id="T23" nextPass="T24">
    <interaction object="theTicketMachine" signature="removeCoins()" type="return">
    </interaction>
</step>
<step id="T24" nextPass="T25">
  <interaction object="theTicketMachine" signature="returnCoins():bool" type="call">
  </interaction>
</step>

<step id="T25" nextPass="T26">
  <interaction object="theTicketMachine" signature="returnCoins():bool" type="return">
    <value>true</value>
  </interaction>
</step>

<step id="T26" nextPass="T27">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
  </interaction>
</step>

<step id="T27" nextPass="T28">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
    <value>true</value>
  </interaction>
</step>

<step id="T28" nextPass="T29">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
  </interaction>
</step>

<step id="T29" nextPass="T30">
  <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
    <value>true</value>
  </interaction>
</step>
<abstractTestSuite>
  <testCase id="T30" nextPass="T31">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="call">
    </interaction>
  </testCase>
  <testCase id="T31" nextPass="T32">
    <interaction object="theTicketMachine" signature="insertCoin():bool" type="return">
      <value>true</value>
    </interaction>
  </testCase>
  <testCase id="T32" nextPass="StatePass">
    <interaction object="theTicketMachine" signature="removeTicket()" type="call">
    </interaction>
  </testCase>
  <testCase id="StatePass">
    <tcStage type="end_tc" verdict="pass"/>
  </testCase>
</abstractTestSuite>