Model Based Testing

A State Machine-based Tool for automated Testing of a Video Conferencing System

Miran Damjanovic

Master Thesis (60 credits)

UNIVERSITY OF OSLO

Department for Informatics

02.06.2009

Preface

Computing is present in many aspects of our lives, often without us even noticing its presence. More complex software systems are being developed all the time, and we are getting more and more dependent on these systems. It is vital for all software systems that they are reliable and robust. Other qualities such as safety, efficiency and security should also be expected from the developed products. This is especially important in critical software systems, such as air traffic controllers for example. The *software testing process* specifies a set of activities that aim at exposing that a system's *intended* and *actual* behaviors do not conform, or to gain confidence in that they do conform. A typical estimate roughly estimates that 50 percent of the total development costs of a software system is spent on testing, including debugging. The testing process is often time and resource consuming, and its repetitive nature is often a motivating issue for the testers. As systems are getting more and more complex they also introduce new challenges considering software quality and reliability. In the ICT industry today the testing processes are often manual, even though there exist automated solutions. These techniques, with high level of automation, are preferable over the handcrafted manual techniques because they are more efficient and less time-consuming. The challenges with automated techniques to software testing are practicality, cost, immaturity and scalability. Despite these challenges, automated solutions for software testing present enormous possibilities in terms of cost-efficiency. It is in this context important to promote research and wider use of automated solutions and tools for software testing, and in this way help establish new and more cost-efficient software testing techniques.

Abstract

Model Based Testing (MBT) exploits the information contained in a *model* in order to derive *test-cases* that can be applied upon the *System Under Test* (SUT). A model describes the states and interactions that a system may be involved in. The models can be derived, sometimes automatically, from the source code of the SUT, or they may have been developed in the early stages of the development process of the SUT. If the models are *formal* (behavior models), a high degree of automation can be achieved. If the models are *semiformal* (or *informal*), we are usually not able to automatically derive test-cases without human intervention. This thesis presents a prototype MBT tool that relies on system specifications, in form of formal and semiformal models, to automatically derive executable *test-cases*. The tool was developed and tested in industrial settings, in cooperation with the Norwegian company Tandberg. The SUT was the software (Video Conferencing System) running on Tandberg's C90 codec (Saturn).The presented solution uses techniques from Model Driven Development (*MDD*), e.g. use of *models* and *model-transformations*, to achieve a high degree of automation. The tool exploits the information contained within a *UML State Machine Diagram*, developed in IBM's *Rational Software Architect* (RSA), in order to automatically generate executable test-cases. The tool was written at Simula Research Laboratory and we tested its performance at Tandberg. The used models were developed locally at Tandberg, and later put in use at Simula. The response from Tandberg was very good, and the testers showed great interest in the tool and the used approach. This thesis presents the applied technique and the technologies used in development process of the described MBT tool.

Acknowledgements

Many people have contributed to the development of this thesis, and the development of the MBT tool presented in this thesis. Among those are the many scientists and researchers that have published papers and books relevant to the topic of my thesis, out of which I have gotten much inspiration. I would like to thank all the people at Simula, especially my friend Tonje, and the Department of Informatics in Oslo for their cooperation and friendliness. I would also like to thank Marius Christian Liaaen from Tandberg for providing the resources that were required during development of the presented tool.

Without the help and guidance of my supervisor, Lionel Claude Briand, and my two co-students, Ph.D. students Hadi Hemmati and Shaukat Ali, this thesis and the developed tool would never have existed. They have helped me with their knowledge and experience in every possible way. Therefore I would like to thank them for their support and understanding.

Finally, I would like to thank my whole family and my beloved girlfriend Ivana for their love and support.

Miran Damjanovic

Oslo, 1. June 2009

Acronyms

ATL	Atlas Transformation Language
CIM	Computation Independent Model
EMF	Eclipse Modeling Framework
IDE	Integrated Development Environment
IRCFG	Interprocedural Restricted Control-Flow Graph
MBT	Model Based Testing
MDA	Model Driven Architecture
MDD	Model Driven Development
MOF	Meta Object Facilities
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
RSA	Rational Software Architect
SUT	System Under Test
UML	Unified Modeling Language
VM	Virtual Machine

Table of Contents

Preface 2
Abstract 3
Acknowledgements 4
Acronyms 5
List of Figures 8
1 Introduction 9
1.1 The context of the thesis 9
1.2 Overall motivation 9
1.3 Structure of the thesis 10
2 The project context 11
2.1 About Tandberg 11
2.2 The testing process and infrastructure at Tandberg 11
2.3 Project with Tandberg 12
3 Background 13
3.1 General introduction to UML and Model-Driven Development 13
3.2 General description of the principles of Model Based Testing 17
3.3 General description of the technologies used 19
3.3.1 Rational Software Architect 19
3.3.2 Eclipse Modeling Framework 19
3.3.3 Atlas Transformation Language 20
3.3.4 MOFScript 24
3.3.5 JPype 27
4 Problem description and objectives 28
4.1 Problem description 28
4.2 Finite State Machines 30
4.3 Generating test cases from UML State Machines 30
4.3.1 State 33
4.3.2 Transition 34
4.3.3 Choice Pseudo State 35
4.3.4 Trigger 36
4.3.5 Guard 37
4.3.6 Effect 38

4.4 Generating Test Data 40
4.4.1 Choosing endpoints 42
4.4.2 The OCL-evaluator 44
4.4.3 Class- and Object diagram in Java 45
4.4.4 Accessing system variables 46
4.4.5 Random parameter selection 47
4.4.6 The setup file 48
4.5 The objectives of the resulting MBT tool 49
5 Tool Architecture 50
6 Relevant technical details 53
6.1 A different approach 53
6.2 My contribution to the project 55
6.2.1 Input models 55
6.2.2 Model transformations 55
6.2.3 Test-data generation 55
6.2.4 Evaluation 56
6.2.5 Example of use 57
6.3 Problem examples and Lessons Learned 60
6.3.1 Problems related to models 60
6.3.2 Problems related to test-case generation 61
6.3.3 Problems related to test-data generation 62
7 Conclusion 63
7.1 Contribution of my thesis 63
7.2 Future work and limitations of the current tool 63
8 Reference list 65
9 Appendixes 66
9.1 Appendix A – SD2IRCFG.java 66
9.2 Appendix B – UML 2.0 Meta-Model (UML) 80
9.3 Appendix C – Transition Tree Meta-Model (EMF) 81
9.4 Appendix D – The domain model of Saturn (UML) 82
9.5 Appendix E – StateMachine2TransitionTree.atl (ATL) 83
9.6 Appendix F - The generated Transition Tree (XML) 89
9.7 Appendix G – SM2TT.m2t (MOFScript) 92
9.8 Appendix H – ClassDiagramTestdata.m2t (MOFScript) 107
9.9 Appendix I – ClassDiagramTestData.java (Java) 113
9.10 Appendix J – TestData.java (Java) 115
9.11 Appendix K – Get_Val.py (Python) 122
9.12 Appendix L – RandomIntGenerator.java (Java) 123
9.13 Appendix M– Sample test-case 124

List of Figures

Figure 1 – MDA, illustration from OMG - [5] 14
Figure 2 – Model abstraction levels 14
Figure 3 – The modeling levels,[7] 15
Figure 4 - Model transformation overview,[7] 21
Figure 5 - MOFScript Preferences 26
Figure 6 – First step of the transformation process, transform a State Machine into a Transition Tree using ATL 31
Figure 7 – Second step of the transformation process, transform a Transition Tree into test-cases using MOF Script 32
Figure 8 - Choice Pseudo State Example 1 35
Figure 9 - Choice Pseudo State Example 2 35
Figure 10 - Specifying triggers in RSA 36
Figure 11 - Specifying guards in RSA 38
Figure 12 – UML Class Diagram of TestData.java 42
Figure 13 - The IEOS Library 44
Figure 14 - Specifying path and sub-path for variables 46
Figure 15 - Tool Architecture 50
Figure 16 – Development activities of the MBT tool 51
Figure 17 – Translating a UML 2.0 Sequence Diagram into IRCFG 53
Figure 18 - Result of test-case execution 56
Figure 19 - Results of evaluating the specified constraints 56
Figure 20 - UML State Machine Diagram 57

1 Introduction

1.1 The context of the thesis

The context of this thesis is Model Based software testing. The Model Based Testing (MBT) tool was developed through a project with Tandberg, described in Chapter 6 – The project context, and its main functionality is automatic test-case generation based on system specifications (models). The developed MBT tool was developed and evaluated during this project. The tool also implements mechanisms that will check for deviations from the states specified in the model (state machine) and the actual system state.

At the core of this tool are model transformations, both a model-to-model transformation and model-to-text transformations. The process of deriving the test-cases can be divided into a set of activities. It has been our goal to automate as many of these activities as possible. Once all the system models are developed and the tool is configured correctly, the generation of the test-cases is fully automated.

1.2 Overall motivation

The motivation and inspiration for developing this MBT tool is a direct result of the need for more reliable and effective software systems. The tool will aim at providing the testers a fairly simple and efficient testing tool that they can use for generating executable test-scripts. The model-based testing approach is far more cost-efficient (especially if automated), and effective, than manual testing. Instead of writing test-cases, the testers will develop models that are used for test-case generation. Another positive side-effect of using models is that the developer's understanding of the SUT will naturally increase. The saved resources (both time and human), in combination with the increased knowledge and understanding of the SUT, justifies the extra effort spent on developing the models. In practice, the testers can use the tool in combination with other testing-techniques, and in this way cover a much larger part of the system domain. The time saved in writing the test-scripts themselves is, in my opinion, reason enough to build, evaluate and hopefully use this MBT tool.

My motivation for writing this thesis comes from the fact that during my studies at University of Oslo I noticed the lack of focus on the software testing process. There exist a variety of Computer Science courses, but none of them address the testing-process specifically.

The testing activities undertaken can generally be summarized to small mechanical tests, such as basic *Unit-testing* (using JUnit [1], for example) and sometimes *Regression testing* while working in groups, and in the development of more complex software systems.

This thesis presented itself as a perfect opportunity for me to gain more experience and knowledge of testing practices, testing-methods and testing tools used in the ICT industry today. It was also very motivating to write a tool that could be put to use in a real world scenario.

1.3 Structure of the thesis

The rest of this thesis is organized as follows: Chapter 2 describes the project context, and introduces our industry partner in this project (Tandberg). Chapter 3 provides some technical background on Model Driven Development (MDD) and Model Based Testing, as well as presenting the technologies utilized by the MBT tool. Chapter 4 describes the problem descriptions and objectives of the MBT tool. This chapter also provides information regarding the generation of test-cases from state machines (UML), and generation and development of test-data. Chapter 5 explains the tool architecture, and Chapter 6 presents relevant technical details. Section 6.1 describes our first solution to model based test-case generation. As later explained, it was decided to abandon this approach, so this section is only relevant as documentation to this completely different approach. Chapter 7 concludes and presents the current limitations of the developed MBT tool and proposed future improvements.

As it may be noticed, the appendix list added to this thesis is somewhat long and consists of several documents. I have deliberately added all the components needed to successfully *generate* the test-cases, as well as components needed to *execute* the test-cases. This is done both to document my own work, but also to ease the understanding of the proposed solution, the technologies used and the structure of the developed MBT tool.

2 The project context

The following sections describe the project context and testing practices used by our industry partner Tandberg. The information presented here is derived partly from conversations with the employees at Tandberg and partly from documents derived in the context of the project.

2.1 About Tandberg

Tandberg is a leading global provider of Visual Communication Systems, with main offices in Oslo and New York. At Tandberg they design, develop and offer systems for video, speech and data transfer. Tandberg has received a number of prizes regarding both design and implementation of their products. This information and the following sections are inspired by Tandberg's official website [2], and also from conversations with employees during my work on the development and validation of the MBT tool.

2.2 The testing process and infrastructure at Tandberg

At Tandberg, a testing team is responsible for running the tests on the developed systems. They write test-cases based on different types of scenarios, which are formulated in a generic TNG script. TNG is an internal language used in the development process as well as for software testing. It is based on Python, and has the possibility of writing in-line Python. This means that one can execute Python code from a TNG script. The drawback is that a Python thread is active in one Python section only. So, for example, in order to have a variable declared in one Python section visible in the consecutive Python section of the test-case, we have to declare the variable in TNG (TNG command *init_var()*), and then when entering another Python section we can fetch the TNG variable (TNG command *get_var()).* The TNG scripts normally consist of a set of calls to/from the test-target and have the "*.ttr*" extension. When the scripts are written, they are uploaded to a server that executes the scripts a FIFO order.

Another possible way to perform a test-run is to manually log on to a Saturn machine, using *Telnet* for example, and invoke calls on other endpoints. Saturn (C90 codec) is a Linux machine having nine child processes. The testing process consists of a series of method invocations on the System Under Test (SUT). For example to make a call from the test-target to another endpoint, one can give the following command in the command-line:

"*testtarget.dial b h323 6b"*.

As the example illustrates we are invoking the *dial* method of Saturn, with first parameter defining the endpoint to call, the second parameter defines which video protocol to use and the third specifies the amount of bandwidth requested.

The testers can experiment with different input values for each of the parameters for all methods defined for a Saturn system, and also invoke methods in varying order.
The testers treat the system under test as a *black-box*, taking an external perspective of the system while developing the test-cases.

The programmers who write the code and implement the software in the final product are testing the code during development, in other words performing white-box testing. They make sure the code is free of errors and implemented correctly. The test department at Tandberg does not deal with testing the source code (implementation) directly; therefore we have left this type of testing out of the scope of this thesis.

2.3 Project with Tandberg

Computing is more and more being present in our everyday lives. More complex software systems introduce greater challenges for *software dependability*. This term addresses the probability that the system will perform its intended functions without degradation. Today, software testing is both time consuming and error prone. Furthermore, software testing is often incomplete due to the lack of automated and scalable software testing techniques. In the ICT industry, the time used at the testing process is naturally limited. There exist commercial tools to support the mechanical aspects of software testing, however the design of test suites from the specification and design information is still not well supported.

The project with Tandberg was undertaken in order to find an automation solution for model-based testing of Tandberg's C90 codec, called *Saturn*. The project itself was divided into a set of activities. In order to automate the testing activities, the design information and specification documents, as well as the *status document*, will be used to derive models. The status document is a XML file containing information about system variables, and is updated every time a change in the system occurs. The models will themselves be used in development of test-suites. This approach to testing is referred to as *Model Based Testing*.

In practice, the resources applied on software testing are often limited, thus identifying the need to focus on the high-risk areas of the SUT. To meet this demand, Risk-driven and Stress-testing techniques will be implemented in the testing process in the future. The project will address the benefits, limitations, costs, challenges and effectiveness of Model Based, Risk-driven and Stress-testing techniques applied on system development projects. Thus, the main goal of the project is to gain deeper understanding of which software testing tasks can be efficiently automated and/or supported.

3 Background

The following subchapters provide some background information regarding the concepts of the Unified Modeling Language (UML) and Model Driven Development (MDD). The material from Section 3.1 are inspired from [3], [4] and [5]. The material on Model Based Testing concepts is in context of [6].

3.1 General introduction to UML and Model-Driven Development

The Unified Modeling Language (UML) is a diagramming notation, and is *the* modeling language for system development. UML is a *formal modeling language*, and the *semantics* of the language are captured in the UML *meta-model*. UML is usually applied to show how the components of a system work together and how the system should meet the customer requirements. Also, the UML is useful when exporting design documents to other workstations/platforms. The use of a known and widely accepted modeling standard helps the developers tackle complexities related to system design and implementation.

The UML can be used in several ways. We can use UML to model small scenarios and develop models that help convey key ideas, but that we can dispose of with later. UML can be used to develop detailed specifications (models) of a software system (or parts of it), using forward/reverse engineering to keep the model consistent with the source code. In the context of Model Driven Architecture (MDA), the UML can be used as a *programming language*, in the sense that the source code is automatically derived from the models.

The core of Model Driven Architecture is Model Driven (software) Development. MDA is a software design approach in which system specifications are expressed as one or another form of models. In other words, it is a model-driven approach to software engineering. MDA separates business and application logic from the underlying platform technology. A model is most often a combination of text and drawings, and is a specification and description of the modeled system and its environment. The models are used in the design, construction, deployment, maintenance, understanding and modification of the software system. Model transformations are used to add detail to and refine model representations. The development of the model transformations requires the developer to be familiar with the business domain as well as the technology being used for implementation. The transformations themselves are then first class entities in the software development process. There are three main advantages by using MDA, namely *interoperability*, *portability* and *reusability*.

A *platform* in the context of MDA is the environment in which the modeled system is supposed to execute. It may be an operating system, but it may also be technology infrastructure (.NET/J2SE and so on) or a specific instance of some hardware topology.

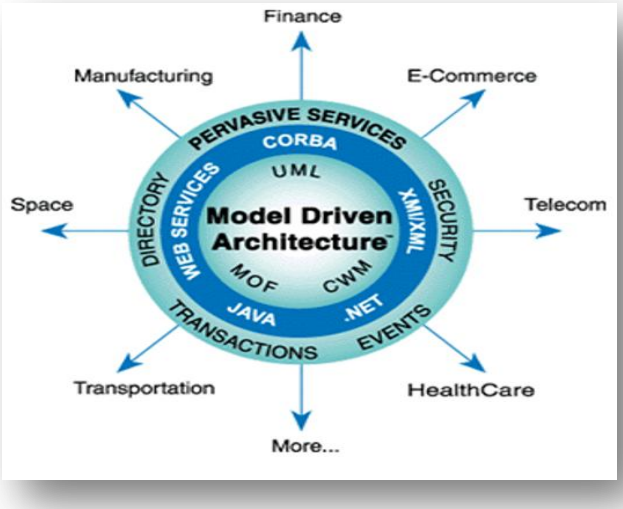


Figure 1 – MDA, illustration from OMG - [5]

There are different levels of modeling abstractions that play an important part in MDA. These are classified according to how well they represent or describe the aspects of the targeted platform. For instance, when specifying business logic, the developers can use a Computation Independent Model (CIM). The main advantage is that the model can be exported to any desired platform that supports the modeling standard used. There are two main such standards: the *Unified Modeling Language* (UML) and the *Eclipse Modeling Framework* (EMF). There exist a great number of both open-source and commercial tools supporting both UML and EMF.

We can separate system specifications from the system implementation and the execution platform supporting the system. Furthermore, we can transform system specifications to specifications relevant for a specific execution platform by specifying the desired software execution platform. The process of exporting a Platform Independent Model (*PIM*) from one execution platform to another is possible because the PIM conforms to its own model, one abstraction level above. This model is called the *meta-model* of the PIM.

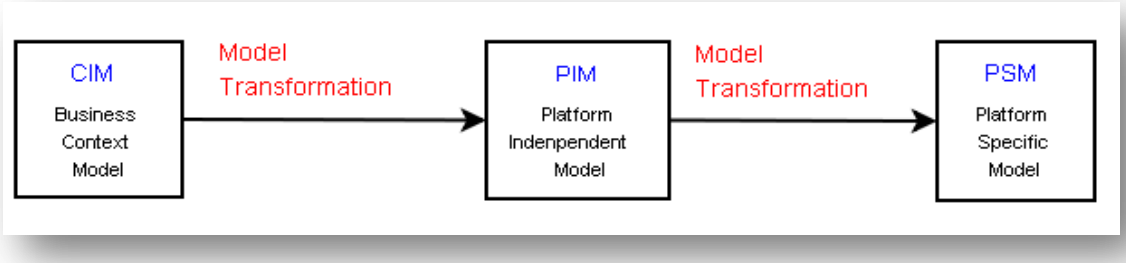


Figure 2 – Model abstraction levels

From Figure 2 we can see that there are three abstraction levels:

1) *Computation Independent Model* (CIM) :
   a. Represents a computational independent viewpoint
   b. Hides the structural and platform-dependant details
   c. Focus on system environment and requirements
   d. Business Context Models

2) *Platform Independent Model* (PIM) :
   a. Represents a platform independent viewpoint
   b. Describes system operation in a platform independent way
   c. Software Specification Models

3) *Platform Specific Model* (PSM) :
   a. Represents a platform specific viewpoint
   b. It is a combination of the PIM and the platform characteristics
   c. Software Realization Models

In order to analyze and have a clear understanding of the semantics of the models inherent to a modeling approach, the models themselves must be described by a model. This model is said to be the *meta-model* of a model. Every model needs to conform to its own meta-model. Following this assumption, there then must be a *meta-meta-model* that describes the meta-model as well. This is indeed true, but the two models are identical, in other words we say that a meta-meta-model defines itself.

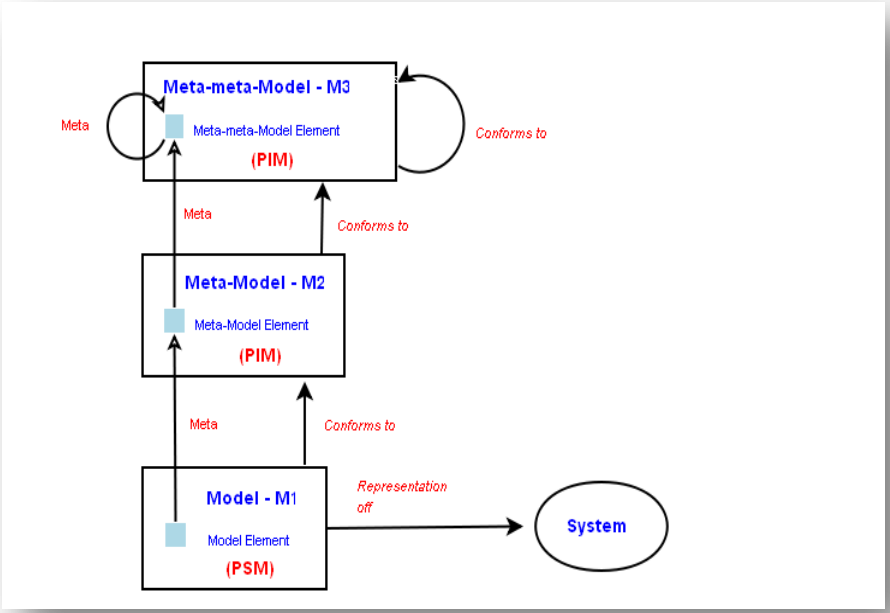


Figure 3 – The modeling levels,[7]

After executing a model transformation, a Platform Independent Model (PIM) can be transformed into one or several Platform Specific Models (PSM), or different PIMs. If the goal of the transformation is to generate PSMs, the PSMs get additional information and grammar inserted by the model transformation, following a code generation template.

3.2 General description of the principles of Model Based Testing

"Model-Based Testing consists in using or deriving models of expected behavior to produce test-case specifications that can reveal discrepancies between actual program behavior and the model." –[6]

Specifications, formal or informal, are a great source of information regarding the structure and fault information of the SUT, making them ideal as a source for test-case generation. Informal system specifications are often expressed in a natural language, making them vulnerable in the sense that they are easily misinterpreted and may be incomplete. Having such specifications as basis for further testing is not ideal because it is difficult to identify errors when it is unclear how exactly the system should behave.

Formal system specifications are specified and modeled using formal methods, based on techniques from logic and mathematics. Such formal specifications and models have more precise semantics, and are more amenable to automatic processing. In this context it is important to differentiate between models derived from the source code and models derived from specifications. Data flow testing approaches to *structural testing*, for example, are based on models derived from the program code. This term for testing, using program structure, is called *Structural testing* or *White-Box testing.*

Model Based Testing or *Functional/Nonfunctional Testing* approaches, on the other hand, utilize *program specifications* in order to derive test-cases. The implementation and design of the program are not important following this approach, as we expect the specifications to contain the information needed for test-case generation. In this context the intended behavior of the SUT is made explicit in the form of *behavior models*. In other words; we do care about the internals of the SUT. The term for testing using program specifications alone is called *Specification-Based testing* or *Black-Box testing*.

The structure defined in a specification may be available to test designers in the form of a semiformal model, Class/Object-diagram, or a more formal model as a finite state machine for example. Deriving test-cases from semiformal models usually involves human judgment, while we may be able to automatically derive test-cases from a formal model. We may also exploit techniques that regenerate models directly from the implementation, if no specifications or models of the SUT exist.

Models help us understand and describe the input space, its structure and sometimes system interactions, and can be used to identify boundaries and error cases when used as an implicit fault model. This can be achieved by intentionally inserting faults in the model. Usually, some form of informal models are available to test-designers as specifications or system requirements.

The cost and effectiveness of model-building should not only be evaluated based on fewer specification errors and possible misinterpretations, but also on the advantages and saved effort in reusing these same models in test design.

Such formal specifications should be considered as important sources of information. This information can be exploited in verifying the tests, testing of the implementation or generation (and/or sequencing) of test-cases. Even though similar research has previously been conducted, both by academic and industrial case-studies, Model Based Testing has still not become a widely accepted industry practice.

The research and development reported in [8] (*TorX*) and [9] (AGEDIS) are most relevant to our work. The TorX test tool is a result of a Dutch research and development project, named *Cote de Resyste*. The tool integrates test-generation, test-execution and test-analysis seamlessly, and is based on the **ioco**-test theory. This tool also integrates test-selection, based on either test-purposes or heuristics. The AGEDIS tools for Model Based Testing are developed through a European Commission sponsored project. The tools provide facilities for modeling, test-generation and test-execution, as well as other test-related activities. The main advantage of these tools is that they can be integrated with other tools from different suppliers, having other requirements and qualities. The AGEDIS set of tools can in this context be viewed as a framework for integration of testing tools. Another approach using mathematical proofs for automated generation and sequencing of test-cases is referenced in [10]. This paper presents techniques for automatic partition analysis and a tool for automatic test-case generation and sequencing, based on state-based system specifications. An evaluation and comparison between Automated Model Based Testing, Manual Model Based Testing and handcrafted/conventional testing is referenced in [11]. The most important findings reported in this paper are that use of models increases the number of detected *requirements* errors, but that the number of detected *programming* errors is independent of the use of models. The derived test-suites detected different kinds of errors, implying that a combination of the techniques could be preferable. Moreover, none of the three techniques that were evaluated detected all errors.

The most important conclusions drawn from these papers are that Model Based Testing indeed can be more efficient than manual testing techniques, but that there often is a problem in applying these techniques to complex systems. It can also be hard, or even impossible, to express the behavior of a complex software system using behavior models. In the cases where a comparison was made with conventional testing techniques, the (semi) automated model-based tools performed *at least as good* as conventional testing techniques. The time spent on deriving the formal models is compensated by more efficient and cheaper testing. This comes from the fact that automated model-based testing techniques tend to produce a greater number of test-cases compared to conventional testing techniques. It is this increase in the number of test-cases that leads to an increase in the number of detected errors. The papers also report on skepticism from the industry partners, related to formal models and the immaturity of the tools.

These tools and techniques are fairly new to the scene, and, in my own opinion, we need these tools to further mature and promote research that will provide us with even better, and more efficient, techniques and tools for Model Based Testing. In this manner we can help establish Model Based Testing as an acknowledged and accepted industry practice in the future.

3.3 General description of the technologies used

In this chapter I will provide some background information and a general description about the technologies that support the functionality of the developed MBT-tool. The three main technologies used are Atlas Transformation Language (ATL), MOFScript and JPype. The ATL-transformation is used to generate Transition Trees from State Machine diagrams, and constitutes the first step of the transformation process. The MOFScript transformation is used in the second step of the transformation process, where the Transition Tree structure is used to generate TNG test-scripts. JPype is used in order to be able to invoke Java methods from within Python code. The Python code here being the section of the test-scripts between the "<%" and "%>.

3.3.1 Rational Software Architect

"***IBM® Rational® Software Architect is an advanced model-driven development tool. It leverages model-driven development with the UML for creating well-architected applications and services.**"* - [12]

Rational Software Architect (RSA) is a professional development and modeling environment, and it is built on the Eclipse software framework. All Eclipse plugins can be imported to RSA and used in the same manner. We used RSA version 7.5 when developing the MBT tool, but currently RSA is been renamed to *Rational Software Architect for WebSphere Software*, starting with version 7.5.

3.3.2 Eclipse Modeling Framework

"***The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model***" - [13]

The Eclipse Modeling Framework (EMF) is used to build tools and applications based on a well defined data-model. Interoperability with other EMF-based tools is also provided. EMF can be imported into RSA (or Eclipse) as a plugin. Several implementations of the OMG modeling standards are present in the EMF as well. The EMF framework includes a meta-model as well, the *Ecore* meta-model. This meta-model is used as basis for describing models and runtime support for the models. Three levels of code generation are provided by EMF. These are the *Model* level including Java interfaces and implementation classes of the model, the *Adapter* level including generation of the implementation classes and the *Editor* level which includes a structured editor for modification (customizing), in conformance with the style of Eclipse EMF model editors.

3.3.3 Atlas Transformation Language

The Atlas Transformation Language (ATL) is developed by the ATLAS INRIA & LINA research group. ATL is specified as a meta-model as well as a concrete textual syntax. The developers generate a number of target models, or just one, from a set of source models. The *rules* in an ATL-transformation specify how the source models are matched and how target model elements are initialized. ATL provides the developers with a set of operations that are used for model manipulation. The ATL-language is a mixture of both *declarative* and *imperative* programming. Besides the model transformation, ATL can be used to specify requirements on the models using model *query facility*, and supports code factorization by making it possible to define ATL *libraries*. So there basically are three kinds of ATL units, namely *transformations*, *queries* and *libraries*. The unit is defined in a distinct ATL-file, and has the *.atl* extension.

I will not go into details about ATL queries, libraries and their usage, as we do not use them in our MBT-tool. The interested reader may take a look at the ATL User Manual [7] , and there are numerous sources of information on the Web about the concepts of ATL. Figure 15 - Tool Architecture depicts our use of ATL for model transformations.

The ATL IDE (Integrated Development Environment) provides the developers with a number of useful development tools that help in design and development of ATL-transformations. These tools are standard developments tools, e.g. a debugger and syntax highlighting, and a number of additional tools for model and meta-model handling. The transformations themselves are executed by the ATL transformation engine. The ATL IDE is built over the Eclipse platform, and can be added to Eclipse as a plug-in. The run-configurations for ATL transformations can be set by using the *Run Configurations* option in the upper left part of the Eclipse, or Rational Software Architect (RSA), window.

As described in Section 3.1, a model is defined according to the semantics of its own meta-model, and we say that this model *conforms to* its meta-model. This means that the elements of a model, as well as their relationships, are defined in the *scope* of the meta-model. ATL is capable of handling meta-models that are either specified in the scope of *MOF* or *Ecore* semantics. MOF is defined by OMG and the Ecore meta-meta-model is defined by the Eclipse Modeling Framework (EMF).

A simple model transformation, like an ATL model-to-model (m-2-m) transformation, will provide operations, such as *rules* and *helpers* in the context of ATL, to generate a target model, **Mb,** from a source model, **Ma**. Furthermore, both models **Ma** and **Mb** have to be defined in the scope of their meta-models, **MMa** and **MMb**, respectively. Note that these two meta-models can be identical, and the model transformation serves as a tool for model refinement in that case. The meta-models **MMa** and **MMb** have to conform to the considered meta-meta-model, MOF or Ecore in the context of ATL.

A key feature in model engineering is to consider all used artifacts, if possible, as models. That means that the model transformation itself, **Mt,** has to be defined as a model, and this model must conform to its own meta-model, **MMt**. This meta-model, the transformation meta-model, defines the model transformation semantics. The figure below illustrates this transformation process.

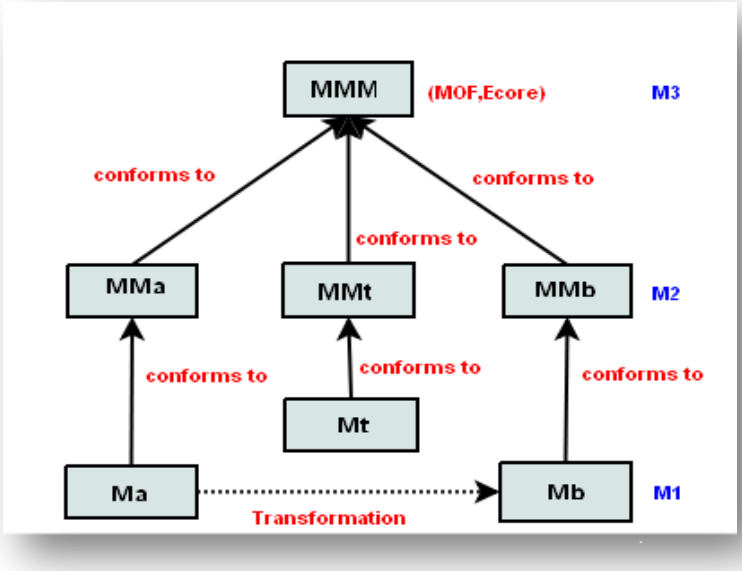


Figure 4 - Model transformation overview,[7]

As mentioned earlier, ATL only focuses on the model to model (m-2-m) transformations which are specified in *ATL modules*. An ATL module is composed of a *header section*, an optional *import section*, a set of *rules* and a set of *helpers*. The rules and the helpers may be declared in any order given certain conditions; they do not belong to any specific sections in the ATL module. The execution of the ATL module is divided into three steps: a) *a module initialization phase*, b) *a matching phase* and c) *a target model elements initialization phase*. The following sections describe the structure of the ATL module.

3.3.3.1 Header section

The header section specifies source and target models as well as the name of the module. The code snippet below, taken from our m-2-m transformation, illustrates a concrete example.

```
module StateMachine2TransitionTree;
create OUT : transitiontree from IN : SM;
```

In the example above we see that the name of the module is *StateMachine2TransitionTree*, the source model (**IN**) is named *SM* and the target model (**OUT**) is named *transitiontree*. We can define more than one source or target model, separating the declared model names by comma, under the condition that their names are unique within the set of models.

3.3.3.2 Import section

The import section of an ATL module makes it possible to declare which ATL libraries are to be imported by the module. The example below is taken from the ATL User Manual [7], as we do not have an import section in our model transformation.

```
uses extensionless_library_file_name
```

The developers may import several different ATL libraries simply by using successive *uses* statements.

3.3.3.3 Helpers

ATL helpers are in a way equivalent to methods in other programming languages (e.g. Java), since they define code that can be called from different places in the ATL transformation. A helper is defined by a *name*, a *context type*, a *return value type*, an *ATL expression* and an *optional set of parameters*. ATL defines two types of helpers, namely *functional helpers* and *attribute helpers*. Both of these types of helpers have to be defined within the context of a data type. Attribute helpers are computed only once, the first time their value is required. A concrete example of an attribute helper is shown below.

```
helper def : initial : SM!Transition =
SM!Pseudostate.allInstances()->select(s1|s1.kind=#initial)->first();
```

The body of a helper is specified as an OCL expression, as the example above illustrates. The code snippet below represents a functional helper and is taken from the ATL User Manual, as we do not use functional helpers in our m-2-m transformation.

```
helper def:averageLowerThan(s:Sequence(Integer),value:Real):Boolean=
let: avg: Real = s->sum()/s->size() in avg < value;
```

The functional helper above returns a Boolean value stating whether the average value of a sequence of integers is lower than a given real value (second parameter). This helper is defined in the context of the ATL module itself, as no context is specified. The context of the helper is specified by the *context* keyword, following the *helper* keyword.

3.3.3.4 Rules

As previously mentioned, ATL provides two different programming modes, namely declarative and imperative programming. As a result, there are also two different types of rules in ATL, and they correspond to the two different programming modes. These two types of rules are the *Matched Rules* and the *Called Rules*.

The matched rules correspond to declarative programming, or the ATL declarative transformation. They specify from which kinds of source models the target models are produced, and specify how the target model elements should be initialized. A concrete example of a matched rule is shown below.

```
rule SignalEvent2SignalReception{
        from
                a:SM!SignalEvent
        to
                b:transitiontree!SignalReception(
                        name <- a.signal.name,
                        qualifiedName<- a.signal.qualifiedName,
                        parameters <- a.signal.ownedAttribute)
}
```

In the example above, a model element (*SignalEvent*) of the SM model is transformed into a *SignalReception* element of the *transitiontree* model. The *source pattern* of the rule is defined by the keyword *from*, and the *target pattern* is defined by the keyword *to*. The target pattern specifies which elements should be generated when the source pattern of the rule is matched. A source model element of an ATL transformation should not be matched by more than one rule.

The called rules, on the other hand, can basically be seen as a special type of helpers. The biggest difference between a matched rule and a called rule is that the called rule does not have to mach a source model element, so they do not have a source pattern. The called rule can nevertheless generate target model elements. Below is a simple example of a called rule, taken from the ATL User Manual, [7], the interested reader may want to take a look at Appendix E – StateMachine2TransitionTree.atl (ATL) where a recursive called rule (*Transition2OutgoingTransition*) can be analyzed for details.

```
rule NewPerson (na : String, s_na: String){
        to
                p: MMPerson!Person(
                        name <- na
                )
        do {
                p.surname <- s_na
        }
}
```

As we can see from the example above, the called rule has no source pattern. That means that the initialization of the target model elements (*Person*) have to be based on module attributes, local variables and parameters. The called rules may also be recursive.

3.3.4 MOFScript

While we use ATL in the first step of our transformation process (UML State Machine Diagram to Transition Tree), we use *MOFScript* in the second step of our transformation process (Transition Tree to TNG test-scripts) as well as for generation of test data (UML Class Diagram to Class and Object diagrams in Java). It is important to distinguish between the MOFScript *Tool* and the MOFScript *model to text transformation language*.

"The MOFScript tool is an implementation of the MOFScript model to text transformation language" – (MOFScript User Guide [14])

The MOFScript Tool provides a set of tools and frameworks that aim to facilitate code generation or documentation from source models. The text-transformation is used to create files, generate code and add detail and grammar to the generated text. The MOFScript Tool currently consists of five components. These are the *MOFScript Lexical Editor, Outline Viewer, Preference Manager, Problem Viewer* and *Result Viewer*. The MOFScript Tool is implemented as an Eclipse plugin. More detailed information about the MOFScript architecture can be found in [15].

The MOFScript language is used to specify a *text transformation* (model-to-text). The text-transformation is constituted out of a set of *rules.* The name of the module is specified using the keyword *texttransformation* followed by a chosen name. The name of the module can be different from the file name, with the file names having the *.m2t* extension. The code snippet below is taken from our m-2-t transformation and shows a concrete example.

……...

```
texttransformation SM2TT (in transitiontree: "http://transitiontree/1.0",
                                    in uml:"http://www.eclipse.org/uml2/2.0.0/UML"){
```

……..

The code snippet above also illustrates the defining of input models (*transitiontree* and *uml*). A text-transformation can define many input models as parameters. A text-transformation can import other text-transformations as well; this is done by using the *import* keyword.

The text-transformation starts its execution at the *entry point rule* (like *main()* in Java).The entry point rules can be specified in the context of a specific data type, but this is optional. If the context type is omitted from the definition of a rule, regardless of the type of rule, the context type is simply the module itself. The context type implies which meta-model element is used in starting the execution. If a context type has several instances, which can often occur when the context type is an element of a model rather that an element of a meta-model, the entry point rule will be executed once for each instance of that type.

The code snippet below illustrates the entry point rule.

……

```
transitiontree.TransitionTree::main(){
        var enumerationList : List;
        uml.objectsOfType(uml.Enumeration)->forEach(e:uml.Enumeration){
                enumerationList.add(e);
            }
```

……

Here we see that the context type of this entry rule is the TransitionTree element of the Transition Tree meta-model, therefore *transitiontree.TransitionTree*. Furthermore, the example illustrates a simple variable declaration and a list instantiation using the *forEach()* iterator.

A part from entry point rules, the text-transformation can consist of several "normal" rules. These rules can also be defined without a context type. The rules can basically be seen as functions, and may define a return type as well as having input parameters. The example below illustrates a rule that is used to find the pre-condition of an UML Operation.

……

```
module::findOperationsPreCond(opName : String){

    var ret : String = " ";
    uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
    c.ownedOperation->forEach(op : uml.Operation){
        if(opName.equals(op.name)){
        ret = op.precondition.first().name;
            }
        }
    }
    return ret;
}
```

……

Variables and properties can be defined both locally within a rule and globally. By creating file(s) we can print the statements we want, and with the syntax and grammar we want, to the file(s). Even if a file is declared in one rule, it can still be the target of output if it is used in a rule that was called from the rule where the file was declared. In other words, as long as the rule where the file was declared is active, the file declaration is active as well. The example below shows a concrete example from our m-2-t transformation.

……

```
file f("TestCase_" + counter + ".ttr");
fileList->forEach(c){
     println(c);
}
```

……

The previous example shows exactly how the text, where the statements are represented as elements in a list (*fileList*), is printed to the files. In our m-2-t transformation we want to generate one file per path of the Transition Tree.

When using MOFScript tool we need to specify the run-configurations, similar to ATL. There are two things that need to be taken care of, given that the MOFScript plugin is installed properly. First we need to specify a set of properties in the MOFScript preferences settings. These are accessed through the command *Window->Preferences->MOFScript Preferences.* The illustration below illustrates the MOFScript preferences.

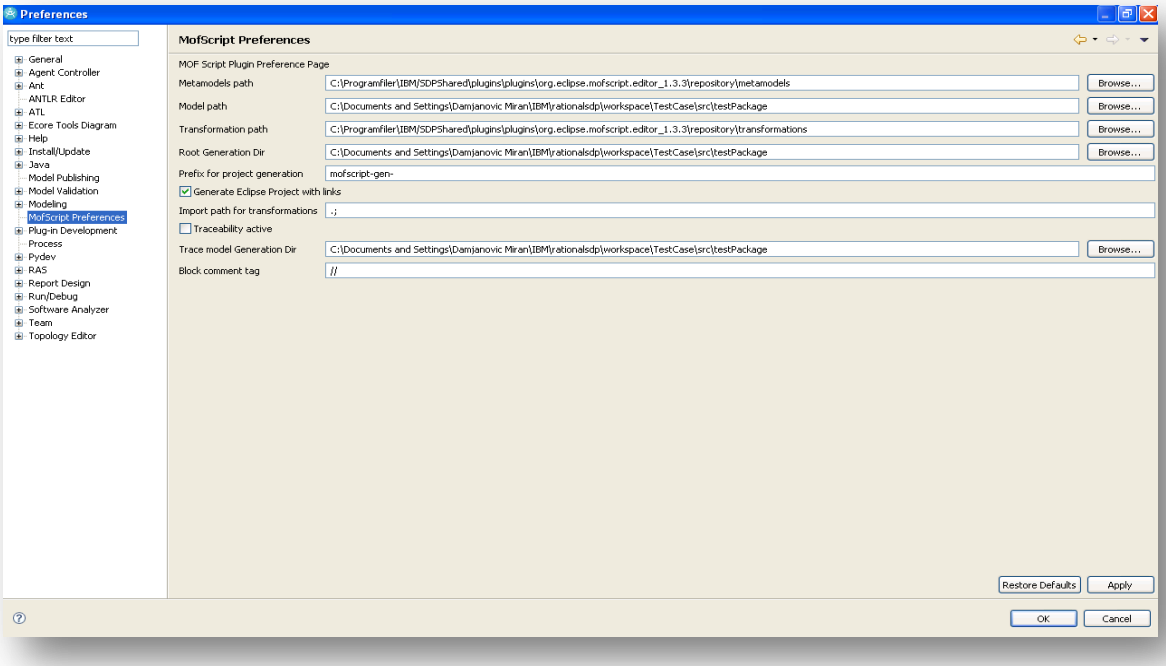


Figure 5 - MOFScript Preferences

Second, we need to add our meta-model to the MOFScript repository, the set of meta-models visible to the MOFScript tool. In RSA, the repository is usually found at the following path:

RSA-directory/plugins/org.sintef.mofscript.editor_1.2.5/repository/metamodels

There is also a growing number of tutorials and user guides available on the Web that explain the details regarding the setup process of the MOFScript tool more thoroughly.

3.3.5 JPype

JPype makes it possible to have full access to Java class libraries from Python code. In contrast to Jython (JPython), this is not achieved through re-implementation, but rather by interfacing at the native level in both Virtual Machines. More information regarding JPype can be found in [16].

We found JPype to be an attractive solution in bridging the gap between the Python code in the produced test-scripts and the test data implemented in Java. We needed JPype in order to be able to call the Java method (*updateObjectDiagram()*) with the OCL expressions and the system variable values as parameters, which updates the Object Model (Java), and returns the results from the OCL evaluator. The method will first update the Object Model, and then execute the OCL query and return the result to caller (Python section of the test-case).

The illustration below shows the usage of JPype in one of our generated TNG test-scripts.

```
 from jpype import *

……

classpath = os.path.join(os.path.abspath('.'))
jpype.startJVM("jvm.dll", "-Djava.ext.dirs=%s" % classpath)
Test = jpype.JClass('Test.TestData')
ClassDiagram = jpype.JClass('Test.ClassDiagramTestData')
t = Test()
c = ClassDiagram()
……

queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls = 0")
……

init_var ('caller', t.getNumberToConnect("testtarget"))
……
shutdownJVM()
```

The code segments above are placed in the first Python part of the test-scripts. The main idea is to declare two variables, *t* and *c,* which will point to a Java object each after calling the constructor of each Java class, *Test* and *ClassDiagram* respectively. We also see how these pointers are used further down in the script. The methods of the object pointed to by **t** return names of endpoints according to the input (“testtarget”), and is thoroughly explained in Section 4.4.1. The method of the object pointed to by **c** updates the Object Diagram and returns a Boolean value as a result of the OCL query on the Object Diagram. The first parameter for the function *updateObjectDiagram()* is the data required for updating the Object Diagram, and the second is the OCL expression of the query.

Even though there are some limitations regarding JPype, e.g. threading, we still found it useful and simple enough for our purposes.

4 Problem description and objectives

In this chapter I will explain in more detail the problems that needed to be solved, and present the objectives of the developed MBT tool. The introduction material and information on the UML 2.0 notation from Section 4.3 are inspired from [17], [18] and [4].

Section 4.1 contains a problem description, Section 4.2 is a short introduction to Finite State Machines, Section 4.3 describes the transformation process that constitutes the core of the MBT tool, Section 4.4 explains more about the generation of test-data and Section 4.5 presents the objectives of the resulting MBT tool.

4.1 Problem description

The main problem addressed by this thesis is the automated generation of test-cases from system specifications, in the form of behavior models (state machines).The automation of generation of the test-cases, based on formal models, can be divided into two aspects. These are the generation of test-cases (Section 4.3) and the generation of test-data (Section 4.4).

We used UML State Machine Diagrams in order to model the behavior of the SUT. This type of UML diagram is used to model the states a system can be in, as well as the events that can cause a change to these states (interactions that a system is involved in). UML specifies two types of State Machines: *Behavioral State Machines* and *Protocol State Machines*. Protocol State Machines are a specialization of the Behavioral State Machines; they show only protocol behavior (e.g. TCP) and are not implementation dependent. We have only used Behavioral State Machines in our work. Our approach uses state machine representations of the SUT as the main input model to test-case generation. The information defined within this model, and its associated domain model, is exploited and used to generate executable test-cases. The domain model is also used for generation of test-data (Section 4.4.3), in correspondence with our goals to automate, as much as possible, the development of the tool components.

The biggest problem regarding the test-data generation was to evaluate the defined constraints at runtime. These constraints are expressed in OCL. We solved this problem by exploiting the features provided by the OCL evaluator. The OCL-expressions are evaluated based on an Object Model that must be constructed prior to the evaluation process itself. The idea then is to have an Object Model in Java representing the current state of the SUT, and querying this model from the generated test-cases at runtime.

The tool itself is built around two main model transformations, which can be seen as two steps of the test-case generation process. The first step (a model-to-model transformation) transforms the state machine (defined in UML) to a corresponding test-model (Transition Tree).

Having the test-model in place, we can now go to the next step executing the second model transformation (model-to-text) and generate executable test-cases. The test-cases are inserted TNG grammar and test-data in the second transformation. The constraints defined in the state-machine are propagated through the transformation processes and eventually evaluated and checked for correctness when the test-case covering the relevant constraint is executed.

4.2 Finite State Machines

"A finite state machine (FSM) or finite state automaton (plural: automata) or simply a state machine, is a model of behavior composed of a finite number of states, transitions between those states, and actions." – (Wikipedia [19])

We can use Finite State Machines to specify interactions between a software component, or a system, and its environment. These *formal models* can provide us with enough information in order to automatically derive test-case specifications. *Semiformal models*, like Object/Class diagrams, usually require some human intervention when deriving test-case specifications. In some cases, the essence of the Finite State Machine is left *implicit*, e.g. expressed in a natural language or grammar, and we can make it *explicit* by deriving a model from the implicit specification of the finite state nature of interactions. Even in systems where the number of states may be infinite, the Finite State Machines are still a useful model for testing and as specification, as the non-finite parts of the specification are possible to derive.

4.3 Generating test cases from UML State Machines

UML State Machine Diagram is a type of *Behavior Diagram* specified in the current UML 2.0 standard [17]. These diagrams, State Machine Diagrams, are also sometimes referred to as State-Chart Diagrams, State Diagrams or State-Transition Diagrams. In the following I will refer to these diagrams as State Machine Diagrams. These types of diagrams are often used to specify the state-dependent behavior of a system, and can be used to derive test-strategies, test-oracles and coverage criteria for software testing. Figure 6 and Figure 7 illustrate our main objectives, and give some pointers to the challenges and problems that needed to be solved.

Our choice of testing method also requires the construction of a test-model, Transition Tree, previously discussed in [20]. The main elements of the Transition Tree are *nodes*, *transitions* and *constraints* derived from UML *States*, *Constraints* and *OutgoingTransitions*, respectively. The meta-model of the Transition Tree is developed in RSA, and can be inspected in Appendix C – Transition Tree Meta-Model (EMF) for a complete definition.

The Transition Tree representation (XML) is derived automatically from the UML State Machine Diagram using a breath-first traversal, implemented in the first model transformation (Appendix E – StateMachine2TransitionTree.atl (ATL)). The second model transformation (Appendix G – SM2TT.m2t (MOFScript)) then traverses the Transition Tree covering each path, and produces code to check the reached states and cause transitions to subsequent states, if possible. The applied coverage criteria, *All Round-Trip Paths* coverage criteria, aims at exposing all incorrect *event*/*action* pairs. This coverage criteria is achieved by covering all transition paths, as described by Binder [21].

In top of Figure 6 we can see a UML State Machine Diagram, developed in RSA. By using transformation rules defined in ATL, Atlas Transformation Language, our objective of this first transformation is to have a representation of the equivalent Transition Tree in XML format. Transitions from UML State Machine Diagrams are mapped to transitions in Transition Tree and states are mapped to nodes in the resulting Transition Tree. The complete transformation can be inspected in Appendix E – StateMachine2TransitionTree.atl (ATL). In this process we also have to eliminate loops and Choice Pseudo states, and we stop the transformation process as soon as we enter a state that has previously been visited.

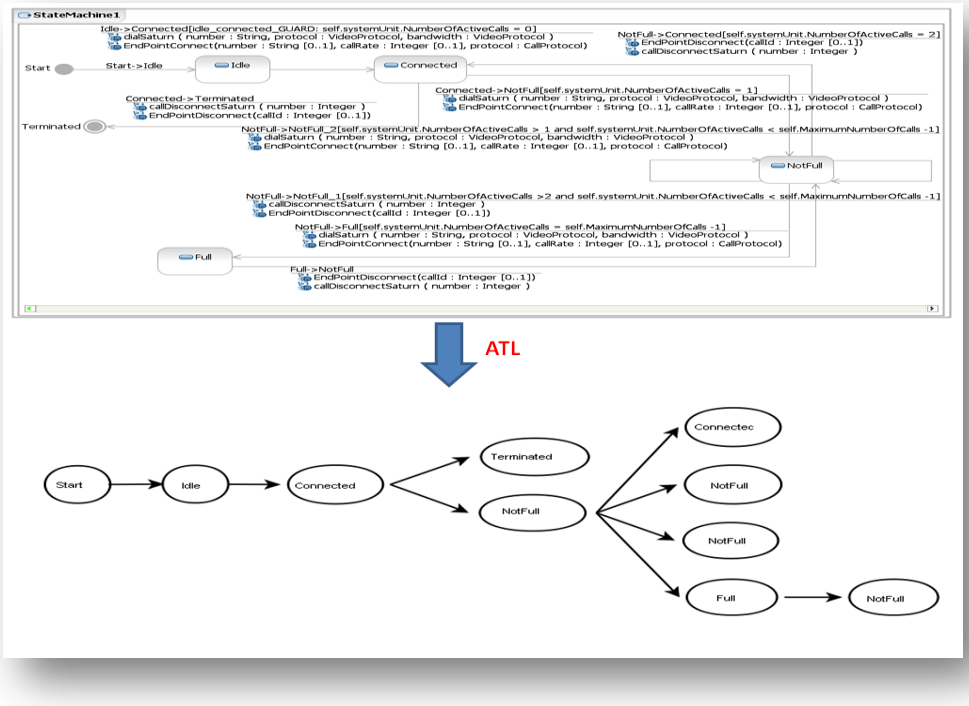


Figure 6 – First step of the transformation process, transform a State Machine into a Transition Tree using ATL

Figure 7 illustrates the second step of the transformation process. In this transformation, the Transition Tree, which was the result of the first transformation, is transformed into executable TNG scripts, with inserted test data and TNG grammar. It is worth noting that this transformation has an intermediate step, namely from Transition Tree to abstract test cases, and then from abstract test cases to executable test-scripts. But we found out that there was no use in having this extra step, as we unfortunately did not have the time required and it was straightforward to do the same process in one step. So we insert the TNG grammar and other test-data all in one step. Each complete path from the Transition Tree will be used to derive one test-case.

At the bottom of Figure 7 is a TNG script, with two method-calls made by the test-target, a Saturn system, namely *dial()* and *disconnect()*. The Python code is omitted from this figure, but I have described shortly what this code does. This is in order to avoid confusion at this point. A whole example of a TNG script can be inspected in Appendix M– Sample test-case.

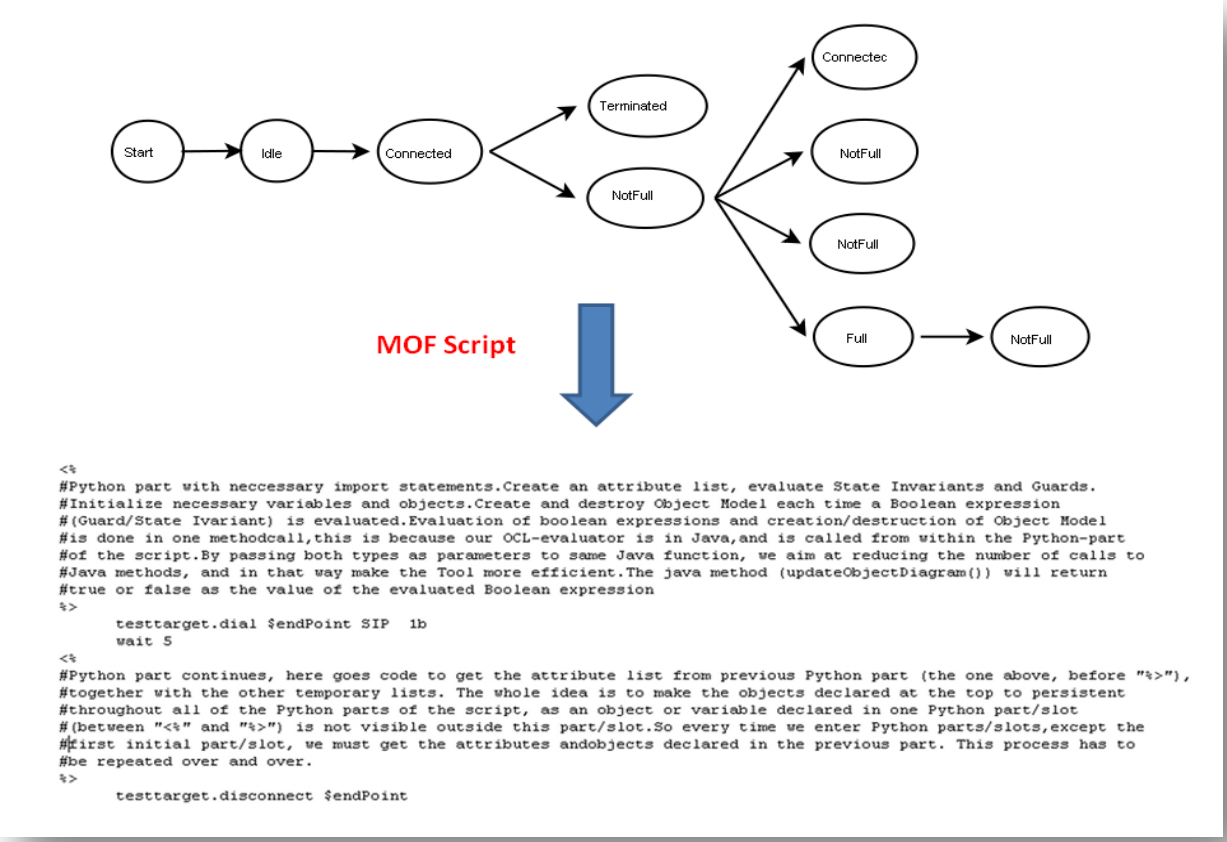


Figure 7 – Second step of the transformation process, transform a Transition Tree into test-cases using MOF Script

The second transformation produces a number of TNG scripts, each one corresponding to one path in the Transition Tree. One path can be expressed as a set of states, omitting the transitions in between for simplicity, for example (*Start*, *Idle*, *Connected*, *NotFull*, *NotFull*). If we inspect the Transition Tree in Figure 7, we see that there are five such paths. This implies that the MBT tool produces five different scripts, each covering a different path in the Transition Tree.

It may seem like there actually are just four different paths in the Transition Tree from Figure 7, but the triggers and guards in the two transitions, both having *NotFull* as both source and target states, are different. This means that these two paths are not unique, even though they bring the system to the same state.

4.3.1 State

A State Machine Diagram describes the states an object or entity may be in. A *state* is expressed as a set of attribute values of the modeled system or unit. A state is modeled as a rectangle with rounded edges. A state has a state invariant (*constraint*), and possibly do, entry and exit actions. When the invariant condition (state invariant) evaluates to true, we can say that we are in a specific point in the behavior of the system, a specific state. When we enter a state, as a result of taking one transition, we say that the state is *active*. Immediately after leaving a state, the state becomes *inactive*. A valid state is expressed using a *state invariant.* The state invariants, as well as other constraints, are defined in OCL 2.0, and are defined based on the *status document* of Saturn.

Do, entry and exit actions are operations that the modeled system may perform while inside, entering or exiting the state in question. A UML State Machine Diagram may be hierarchical, meaning that some states may be *Submachine states* or *Composite states*. Composite states are also called *Orthogonal states* and can have several regions as sub-states. Submachine states are similar to Composite states except that they are intended to group states into a *Sub-State Machine* in order to be able to reuse them at a later point.

It is worth noting that the MBT tool developed, at its current version (May 2009), only can handle flattened State Machine Diagrams, non-hierarchical UML State Machine Diagrams developed in RSA. Another tool currently being developed at *Simula Research Laboratory* is aiming at flattening hierarchical UML 2.0 State Machine Diagrams. This tool can be applied at a hierarchical UML State Machine Diagram, the resulting flattened UML State Machine Diagram can then be given as input to our MBT tool.

In our first step of the transformation process, ATL-transformation, one instance of a state defined in the UML State Machine Diagram is mapped to one or several instances of a *Node* in the produced Transition Tree, the number is dependent on the number of paths in the produced Transition Tree that cover the relevant state.

The produced Transition Tree, together with the domain model (UML Class Diagram) is used as input to our model-to-text (*m-2-t*) transformation. The rules defined in this transformation produce scripts that contain code to verify that a state of the system matches the expected state, defined in the State Machine Diagram, at runtime. For example, the state invariant of the *Connected* state is expressed in OCL in the following manner:

```
self.systemUnit.NumberOfActiveCalls=1 and self.conference.PresentationMode = 'Off' and
self.conference.calls->select(c:Call | c.incomingVideoChannel->asSequence()->last().Protocol <>
VideoProtocol::Off  or c.outgoingVideoChannel->asSequence()->last().Protocol <>
VideoProtocol::Off) ->size() =0
```

The produced scripts have implemented mechanisms that fetch the system variable values of variables that are needed to define a state (Section 4.4.4), and also mechanisms that will check if that state matches the expected state (Section 4.4.2 and Section 4.4.3), defined in the state machine.

4.3.2 Transition

In order to go from one state to another, a *transition* is required. In UML, a transition is modeled as an arrow, with the tip pointing to the target state, and describes the relationships between states and/or *pseudo states*. There are four different types of transitions defined by UML, and there are no specific symbols associated with any of these types. As we will only be using flattened State Machines, the only transition-type relevant for us is *Internal Transition*. Transitions of this type describe the relationships between states belonging to the same composite state, meaning that a transition of this type is not allowed between two different regions within the same composite state. All transitions shown in models throughout this thesis are Internal Transitions, and I will refer to them only as *transitions*. A transition may be associated with a *guard*, *trigger* and *effect*.

The guard is the predicate associated with the event on the transition, the trigger is the event (either signal reception or API call in our tool) and the effect is an operation that will be invoked as a response of taking the transition.

A transition defined in the UML State Machine Diagram is mapped to one or several instances of a *Transition* in the produced Transition Tree, the number depends on the number of paths in the produced Transition Tree that cover the relevant transition.

4.3.3 Choice Pseudo State

A *pseudo state* is a particular kind of a state describing special behavior between normal states, and is usually used to express more complex logic in state changes relative to a State Machine [18]. The Choice Pseudo State has more than one outgoing transition, each with associated unique constraints (guards). The example below illustrates an example of a Choice Pseudo State having two outgoing transitions, modeled in RSA.

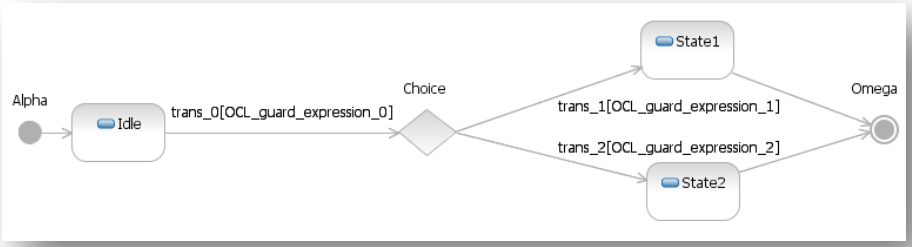


Figure 8 - Choice Pseudo State Example 1

In order to be able to express the same semantics in our resulting Transition Tree representation, we need to merge the two outgoing transitions (*trans_1* and *trans_2*) with the incoming transition (*trans_0*) of the Choice Pseudo State. The associated guard expressions are concatenated using the operator *and.* The example below illustrates the result of this procedure applied to the State Machine Diagram from Figure 8.

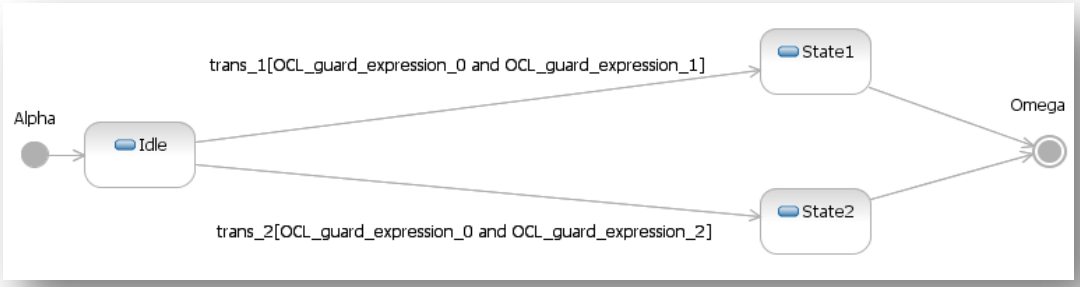


Figure 9 - Choice Pseudo State Example 2

The small state machine from Figure 9 shows how the state machine from Figure 8 is thought of being defined in the corresponding Transition Tree. As we can see, the guard expressions are merged and the Choice Pseudo State is eliminated.

4.3.4 Trigger

A transition may be triggered by an internal or external event, relative to the modeled system. UML specifies four different kinds of events, namely *Call Events*, *Change Events*, *Signal Events* and *Time Events*. Instances of these events are modeled as *triggers* for one or several transitions. This is in practice done by creating a UML Class Diagram (domain model) of the system under test. Then, while developing the UML State Machine Diagram we can reference the elements of these classes, such as methods and attributes, and also signals.

In our MBT tool we were only concerned with the Call Events and the Signal Events, since our test generation strategy required only these two types of events. The example below illustrates how we have defined two triggers for the transition from state *Idle* to state *Connected* in RSA, from Figure 6.

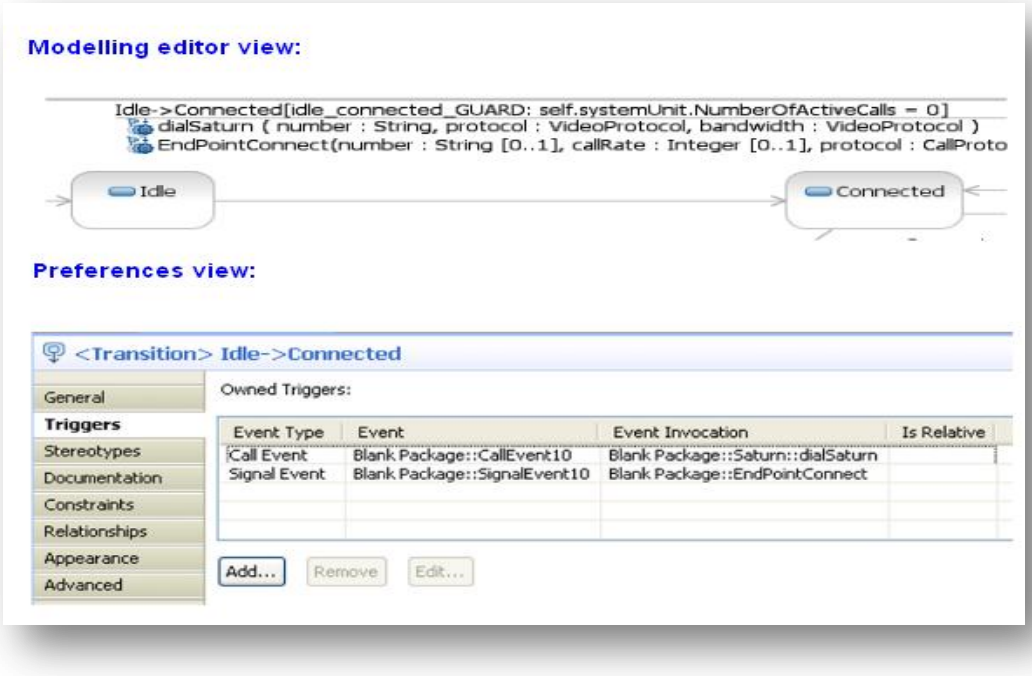


Figure 10 - Specifying triggers in RSA

One instance, of either Signal Event or Call Event, is mapped to one instance of either *SignalReception* or *MethodCall* in the produced Transition Tree, also here the number will depend on the number of paths that cover the triggers. Both *SignalReception* and *MethodCall* from the Transition Tree have the class *Trigger* as their super-class. If several types of triggers are specified for one transition, one of them is picked at random. If the type of trigger is *MethodCall*, the test-target itself initializes the API call, if the type is *SignalReception*, another endpoint will be picked from the pool of endpoints and that endpoint will initialize the API call.

To illustrate the previously described approach, I have provided code samples from two test-scripts, produced by two independent runs with the MBT tool, that show how each of the triggers from Figure 10 are transformed into executable API calls.

(1) *MethodCall (dialSaturn())*:

```
…….
init_var ('caller', t.getNumberToConnect("testtarget"))
%>
        testtarget.dial $caller SIP  2b
        wait 5
<%
…….
```

(2) *SignalReception (EndPointConnect)*:

```
…….
init_var ('caller',t.getCaller("dial"))
%>
        $caller.dial testtarget H323  1b
       wait 5
<%
…….
```

The *getCaller()* and the *getNumberToConnect()* methods are implemented in the TestData (.java) component. This component, along with the generated *ClassDiagramTestdata* component (representation of Class and Object diagrams in Java, Section 4.4.3), is combined in one Jar file (TestDataClasses.jar) and in this way accessible from all generated test-scripts.

4.3.5 Guard

A guard is a Boolean expression, and unless this expression is evaluated to *true*, the associated transition can never fire. In RSA, we can define guards using OCL. An instance of a guard defined in the UML State Machine Diagram is mapped to one instance of a *Guard* defined in the produced Transition Tree. The transitions with associated guards have all associations to this guard in the produced Transition Tree, so here we have a one-to-one mapping between guard from UML and guard in Transition Tree.

In the generated test-scripts the guard predicates are evaluated at runtime, and the resulting Boolean values are stored. The evaluation of the guard expression is done in the same manner as the evaluation of state invariants. This approach consists of first finding out which variables are used in the guard expression. Then we get the current values of these variables from the system (using Get_Val component, Section 4.4.4). The last step is to update the Object Model with these values and execute the query (evaluation). All the evaluation results are stored in the same array, and printed to the screen at execution end.

The picture below illustrates how we can specify guards on transitions in RSA, and is in context of the example transition from Figure 10.

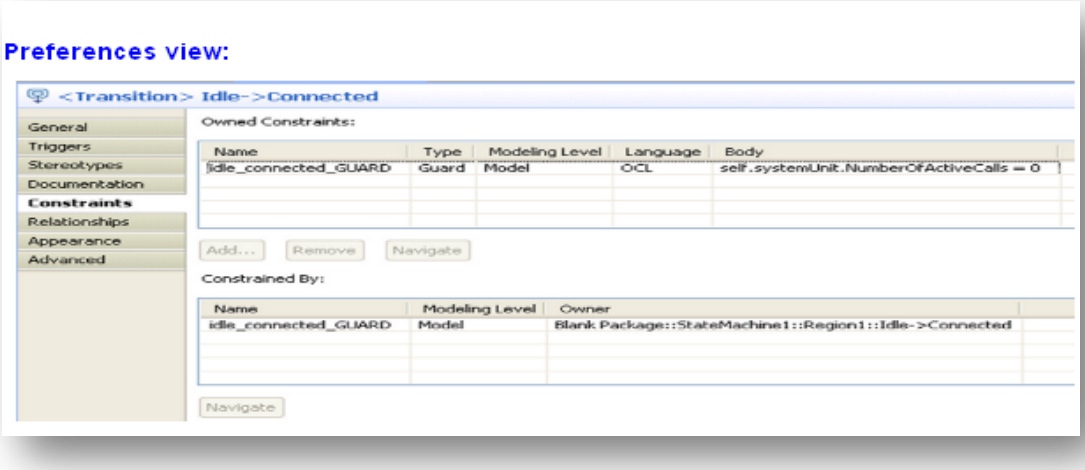


Figure 11 - Specifying guards in RSA

4.3.6 Effect

An *effect* is an action that will be invoked as a consequence of taking the associated transition. The effect is usually a generation of events, such as Signal Events and Call Events (method invocations). For example, in the specification of the effect we can set a name of a method from the mapped UML Class Diagram (domain model), the effect of the associated transition will then be that this method will be invoked.

In the current implementation of the MBT tool it is possible for the user to set the specification of an effect to either a method call or signal event. This way the user may specify to invoke a specific method or generate a signal event as an *effect* of taking a transition.

If the domain model (UML Class Diagram) is *broad* enough, we can define whichever method we find useful/important as being an event on an effect, where the effect is associated with a transition. This approach can be used in *Stress testing*, for example. Methods can be specified in this manner in the *do*, *entry* and *exit*-actions, associated with each state, as well. These actions can all have associated *pre-conditions* and *post-conditions*. These are predicates which must be evaluated to *true*.

The transitions with an associated effect have associations to this effect in the produced Transition Tree, and there is a one-to-one mapping between an effect from UML and an effect in the Transition Tree representation.

The previous sections describe the most important elements from UML State Machine Diagrams that are used in the MBT tool. Other elements are transformed in a similar manner, and are defined within the elements previously described. These are elements like actions (entry/exit/do-actions that can be associated with states) and parameters.

More detailed description and information about UML State Machines and other diagrams from the UML family can be found in [4] and [18]. The interested reader may want to take a look at the developed meta-model for Transition Trees, which is added as Appendix C – Transition Tree Meta-Model (EMF) to this work. The relevant portion of the UML 2.0 meta-model is added as Appendix B – UML 2.0 Meta-Model (UML). The meta-models will provide more insight and detailed information on the structure of the input models required by the MBT tool.

The two model-transformations constitute the core of our MBT tool. The transformation rules defined within those two transformations are fairly straightforward to understand, and are added as Appendix E – StateMachine2TransitionTree.atl (ATL) and Appendix G – SM2TT.m2t (MOFScript).

4.4 Generating Test Data

The developed MBT-tool requires six additional components, in addition to the two main model transformations, in order to function properly. These components are commonly referred to as *Test Data.* Four of these components are hand-written, one is a Java library and the last component is derived automatically from the domain model (Class Diagram) using model transformation. This chapter describes these components, their function and their utilization.

Test Data consists of the following components:

1) A Java class with methods that will, according to the input, return a name of an endpoint that will either be called/disconnected from the test-target (SUT) or call/disconnect the test-target itself. The complete source code of this component may be inspected in Appendix J – TestData.java (Java).

2) In order to evaluate the OCL-expressions, such as guards and state invariants from the state machine, we use the IEOS Java library. This library is packaged as a Jar and is utilized *(imported)* by the automatically generated java class that contains methods to create Class- and Object diagrams given the Class Diagram as input, and execute a *query.* The Class- and Object diagrams are needed because the query, containing the OCL-expression, is evaluated based on the instantiated Object diagram (in Java).

3) The OCL-evaluator from 2) is used by objects that are instances of an automatically generated Java class. This class can be inspected in Appendix I – ClassDiagramTestData.java (Java). The purpose of this object, which has just one method, is to instantiate an Object diagram such that the OCL – evaluator (IEOS-Library) may be queried, as the results of the OCL-expressions are based on the current state of the Object Model. The model transformation that generates this Java class can be inspected in Appendix H – ClassDiagramTestdata.m2t (MOFScript). The input to this transformation is the domain model, UML Class Diagram, used in the other model-to-text (MOFScript) transformation as well.

4) In order to make the Object Model from 3) consistent, we need to have a function that will take a path to a system variable as input, and return it's value as result. This value is then inserted in the Object diagram and then the query is invoked. This procedure is repeated for every variable found in an OCL-expression. A problem with this solution is that every time we query the OCL-evaluator we need first to instantiate the Class- and Object diagrams, which can be costly if the expressions are large and numerous. The code is added as appendix to this thesis, and can be inspected in Appendix K – Get_Val.py (Python).

5) This component is a simple Java class that has one method which returns a randomly generated integer number. We needed this random number generator to pick the parameter values and triggers randomly (if the user specified more than one trigger per transition). The MOFScript does not facilitate random number generation, so we invoke the Java method that produces the randomly generated integer from within the transformation rules. The Java class is added as Appendix L – RandomIntGenerator.java (Java).

6) The IP addresses of the endpoints that will be used during testing must be specified in a *.tts* file. This is just a simple text file which contains the names of the endpoints and their IP addresses.

4.4.1 Choosing endpoints

At the point at which the test-scripts are generated we do not have any knowledge of the endpoints that are going to be used in the actual test-execution. It would also be impractical to require the user to specify this information during model transformation. As an effect we need some kind of mechanism that will give us the name of the endpoints by parsing the .tts file at runtime. The solution we used is to have a Java class with some functions that will do this, and return the name of an endpoint according to the input. The UML Class Diagram of this component is shown in Figure 12 – UML Class Diagram of TestData.java.

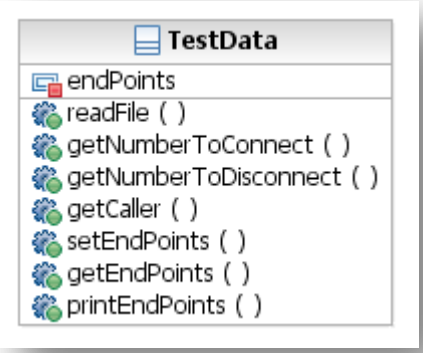


Figure 12 – UML Class Diagram of TestData.java

By taking advantage of *JPype* we can invoke functions of objects that are instances of this class during runtime. The object that is an instance of this class will also keep track of the connected endpoints so that one does not initialize a call to an already connected endpoint. There are two situations to consider. The first is where the test-target will initialize a call or a presentation, a *MethodCall*. In that case we will give "testtarget" as input to the *getNumberToConnect()* method, and the function will return an available endpoint, if any.

An example of the first scenario:

```
…..

init_var ('caller', t.getNumberToConnect("testtarget"))
%>
        testtarget.dial $caller SIP  3b
        wait 5
<%
…..
```

Here, we initialize a variable *caller* with the value returned by the *getNumberToConnect()* function. The *init_var* function is used to initialize variables in TNG. Then the test-target will make a call to the endpoint that is given as *caller*.

In the second case, we want an endpoint to call or disconnect from the test-target, a *SignalReception*. We then use the *getCaller()* function that will, according to the command given as input, return an appropriate endpoint. What an *appropriate* endpoint is, is decided both by the input command, which may be *dial*, *disconnect*, *openduo* and *closeduo*, and by the current state of the SUT.

An example of the second scenario:

```
….

init_var ('caller',t.getCaller("disconnect"))
%>
        $caller.disconnect testtarget
        wait 5

<%
….
```

As with the first scenario, we initialize the variable *caller* except that this time we invoke the *getCaller()* function with *disconnect* as input, as this is the name of the function that the endpoint is going to invoke. After the variable is initialized, we enter the TNG part of the script and the endpoint will then disconnect from the test-target.

4.4.2 The OCL-evaluator

We use OCL to express states (state invariants) and guards on transitions in the UML State Machine Diagram. These expressions must be evaluated to true/false during runtime. One solution would be that we wrote some kind of component that does exactly this. We did not go for this approach as it would be fairly complicated to do in the time that we had, and we also came across the IEOS Java Library that can do just this. The used approach (IEOS) is further explained in this section. A Class Diagram for the IEOS library is shown below.

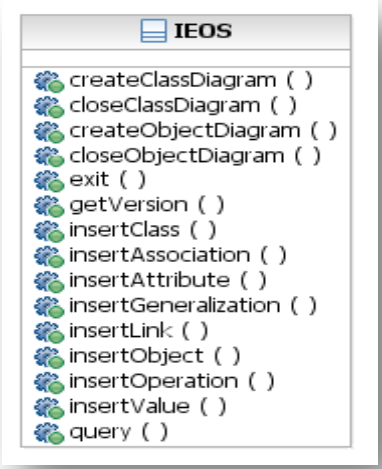


Figure 13 - The IEOS Library

A query is constructed by giving the name of the context and an invariant, in addition to the OCL-expression. Since out test-target will be the C90 codec, Saturn, the complete string given to the *query()* function will be as follows:

"context Saturn inv name: the OCL-expression"

The following example illustrates this:

ieos.query("context Saturn inv name: self.systemUnit.NumberOfActiveCalls=0")

The query is evaluated based on the Object diagram that is constructed in Java. The result of the query is returned to the TNG-script and logged. We do not use these values for anything other than inspection at the current time of writing.

One possibility is to write a *Test Oracle* that will pass or fail scripts according to these evaluations of OCL-expressions. Another possibility would be to integrate these results in the logic. For example they could serve as condition statements (if/else), and if the condition was not satisfied we would not follow that path. This would mean that if a guard expression was evaluated to false then the *MethodCall* or the *SignalReception* would not be invoked. There does not exist such a mechanism in the current version of the tool.

4.4.3 Class- and Object diagram in Java

In order to take advantage of the OCL-evaluator from the previous section we need to create a UML Class Diagram and initialize an Object diagram in Java, based on which we can construct our queries. The Class Diagram needs to be consistent with the SUT and the Object diagram needs to be updated each time a change in SUT occurs. As I mentioned earlier, the data initialized in one Python section are not visible once the execution enters another Python section of the TNG-script. So, we need to create the Class- and Object diagram each time we want to evaluate an expression. In order to keep the amount of updates to the Object diagram as low as possible, the expression that is to be evaluated is parsed. During the parsing we collect all of the variable names present in the expression. Once we get the exact values of these variables from the system, using the *Get_Val* component, we update the Object diagram with the correct values, and then initialize the query. The Java class containing the Class- and Object diagram in Java is generated automatically using model transformation, MOFScript. The only input to this transformation is the sameUML Class Diagram used by the m-2-t transformation that generates the test-scripts. This is in line with our strategy to automate as much as possible regarding the tool components. A code snippet from the generated file is shown below.

```
package Test;
import core.IEOS;
import java.util.*;
public class ClassDiagramTestData {
private IEOS ieos;
public ClassDiagramTestData(){
}
public String updateObjectDiagram(String updateData, String   queryValue) throws Exception{
        ieos = new IEOS();
        ieos.createClassDiagram();
        ieos.insertClass("Saturn");
        ieos.insertClass("TNG");
        ……
        ieos.closeClassDiagram();
        ieos.createObjectDiagram();
        ieos.insertObject("Saturn","saturn");
        ieos.insertObject("TNG","tng");
        ……
        String query = "context Saturn inv name: " + queryValue;
        try{
                System.out.println("Executing Query: " + query);
                queryResult = ieos.query(query);
        }
        Return queryresult;
```

To reduce the number of calls from the test-script to this external component, we update the Object Diagram and initialize the query in one go as the example above illustrates. First we create the UML Class Diagram, then we populate the Object diagram and finally we query and return the result to the TNG-script.

4.4.4 Accessing system variables

In order to update the Object diagram from Section 4.4.3, we need to somehow get the actual values of the system variables, and then use these to update the Object diagram. Our solution was just to have a simple Python function that will return the current value of a variable(s) given its path as input. The function was written by an employee at Tandberg in cooperation with myself. The path and sub-path give the precise location of the variable, and they are specified in the UML Class Diagram as shown below:

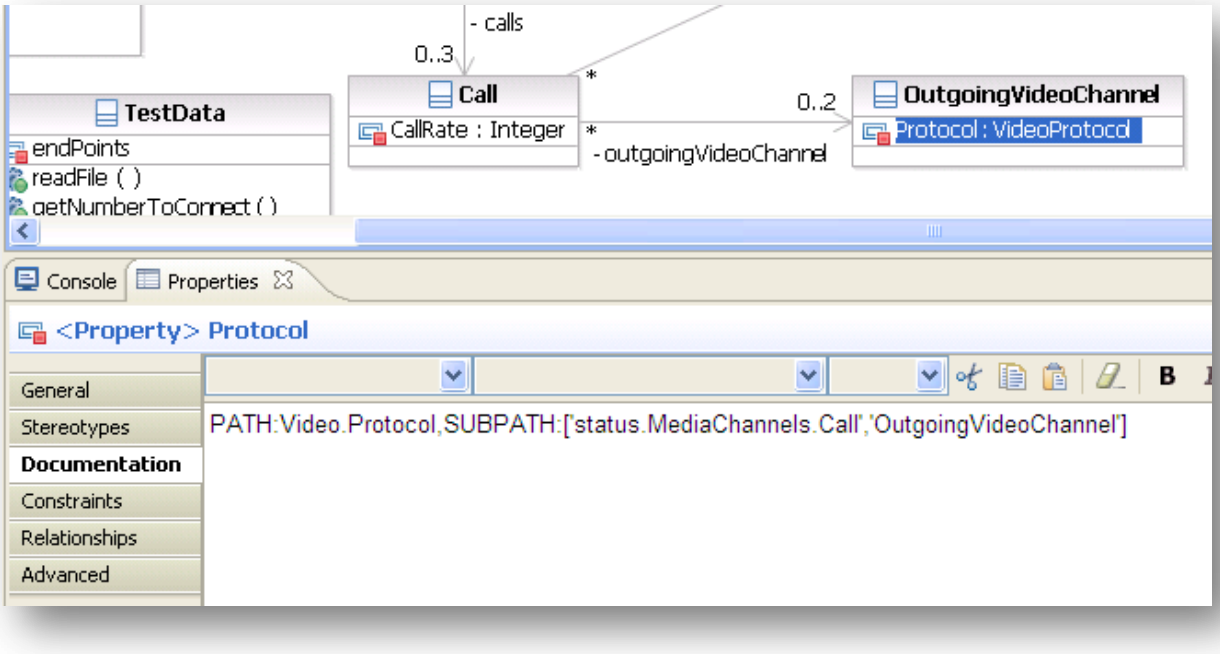


Figure 14 - Specifying path and sub-path for variables

In the figure above we can see that the path of the variable *Protocol* of the *OutgoingVideoChannel* is set to "Video.Protocol" and similar is done for the sub-path. This information is the input to the Get_Val() function that will return the value of the variable given its path and sub-path.

The complete code for this component is added as an appendix, and can be further inspected in Appendix K – Get_Val.py (Python).

4.4.5 Random parameter selection

One of the most important facilities of the MBT tool is to randomly select parameter values, from a *pool*, when generating the test-scripts. The parameters and their types are derived from the input models, Class Diagram and Transition Tree diagram, during the m-2-t transformation. The parameter types are of types *String*, *Integer* and *Enumeration*.

Unfortunately, we could not find any features of MOFScript that have the ability to randomly generate integer numbers. The solution was to have a Java class with just one method that will do exactly this. MOFScript has the possibility of allowing calls to Java methods from within the transformation rules. An example is shown below, and illustrates the case when the type of the parameter is Integer:

```
if(p.type.name.equals("Integer")){
  generatedInteger = java ("RandomIntGenerator","getRandomInt",3,javaPath);

 param += "  " + generatedInteger;
```

A similar approach is used when the type of the parameter is of type Enumeration. In this case we have a collection of values to pick from, namely the values specified in the UML Enumeration (Class Diagram). The example below illustrates this scenario:

```
index = enumerationLiteralList.size();
index = java ("RandomIntGenerator","getRandomInt",index,javaPath);
        temp = 0;
        enumerationLiteralList->forEach(l : String){
            if(temp==index){
                param = l;
                break;
            }
            temp+=1;
```

The last possibility is that the type of the parameter is of type String. In this case we do not need this random number generator. The reason for this is that parameter values, for the parameters of type String, are derived from some other function. This is specified in the Class Diagram for every parameter, of type String, belonging to some function of the TNG class.

4.4.6 The setup file

The setup file, having the .tts extension is crucial to the test-execution as well as for our own purposes. It is normal to have the names and IP-addresses of the endpoints that are going to be used in a small text-file. This file is given as input in the command line when executing the test-scripts. The following example illustrates this:

```
…> tngrun  test.tts TestScript_0.ttr –l c:\logFile.txt
```

The setup file itself looks like this:

```
a.ip: 10.47.23.21
b.ip: 10.47.23.20
c.ip: 10.47.19.92
testtarget.ip: 10.47.23.54
```

We also use the setup file to get the names of the endpoints. This is done in the TestData.java component. As mentioned earlier, this component will return the names of the endpoints according to the given input, see Section 4.4.1. The names of the endpoints are collected from a setup file similar to the one presented above.

4.5 The objectives of the resulting MBT tool

The main objective of the resulting MBT tool is automated test-case generation, based on behavior models (UML State Machine Diagram). This objective can be further decomposed into two sets of activities. First, and most important, is that a set of test-cases (test-scripts), or just one (depending on the complexity of the UML State Machine Diagram), are generated correctly. Second, the results of the evaluated constraints should be collected and reported. As I mentioned earlier, we define four kinds of constraints. These are the state invariants, guards and pre/post-conditions associated with API calls and actions.

The first objective of the MBT tool is achieved by executing the two main steps of the transformation process. First we generate a Transition Tree diagram from a State Machine Diagram by using ATL. Second, the produced Transition Tree diagram is used as input to the second transformation (MOFScript) along with the domain model (UML Class Diagram), which generates the test-cases, one for each complete path in the Transition Tree diagram.

The second objective is achieved by generation of test data, described in Section 4.4. The test data components provide facilities that make it possible to evaluate the OCL expressions and collect the result at runtime. These components can be seen as building blocks of a primitive type of a test oracle, even though this oracle does not fail or pass any test cases without human intervention at the time of writing. The results are collected in a sequence, which can serve as basis for passing or failing the test cases, as the results of the evaluation of OCL expressions are Boolean values e.g. true/false.

Combining these two objectives, the purpose of the MBT tool is to provide an efficient and automated testing solution for testing of the Tandberg's C90 codec, Saturn. IBM's Rational Software Architect serves as a modeling and development platform, where all the artifacts (modules/files) that are required in the testing process can be developed. The somewhat expanded meta-model of Transition Tree, added as Appendix C – Transition Tree Meta-Model (EMF), contains enough details to be able to undertake different types of testing, e.g. Stress testing, Robustness testing and so on.
The advantage of having an automated testing solution lies in the amount of saved time in writing the test-cases. In the ICT industry, the time devoted to software testing is frequently limited. In this context we can conclude that in order for the MBT tool to be an attractive approach for software testing, the tool needs to outperform the manual approach in the amount of time needed to produce the test-suites. Having provided the domain model (UML Class Diagram) and a state machine (UML State Machine Diagram) the generation of test-scripts is done in a matter of seconds, depending on the complexity of the state machine. Even for the most complex cases that we tried out, the amount of time is at its minimum compared with the manual approach. Another advantage is that we, as far as it is possible, have eliminated human involvement in deriving the test-cases, saving time on spelling errors and other similar human-produced errors. Also, by working with models of the SUT, the developers will have an abstract view of the SUT, increasing the developers understanding of the structure and interaction semantics of the SUT.

5 Tool Architecture

The tool architecture is depicted in Figure 15. I have described only the most important components of the tool in this model, which illustrates the flow of information as well. It was necessary to combine structure with execution semantics in this diagram, as to give a complete overview of all the files and artifacts that are associated with the MBT tool. The reader familiar with ATL will notice that the two input meta-models to the ATL-transformation are omitted.

The two meta-models, UML 2.0 and Ecore, are specified in the run-configurations of ATL. The UML 2.0 meta-model is available through the EMF-registry in RSA, and we have developed the Transition Tree meta-model according to EMF. As the meta-models, once verified, are less likely to change I have omitted them as input elements from this model. The reader should be aware that these two meta-models, in addition to the source model and a name for the target model must be specified in the run-configurations of the ATL-transformation in order for the transformation to run correctly.

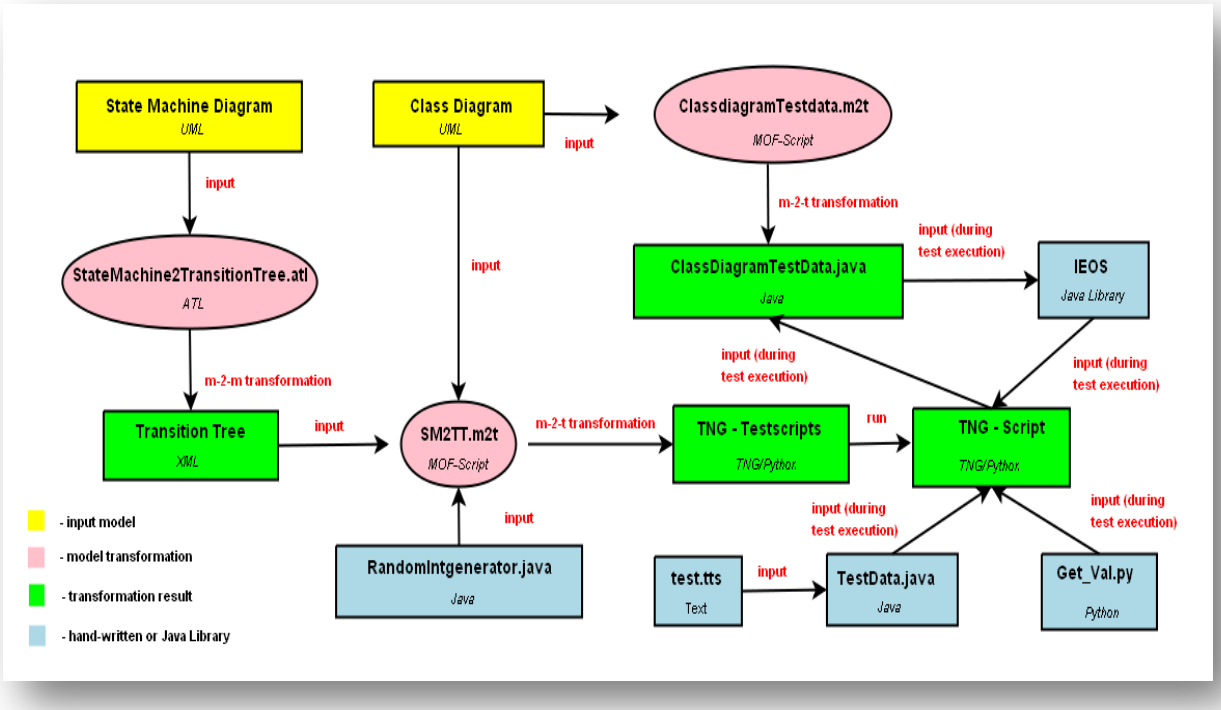


Figure 15 - Tool Architecture

The component diagram above suggests that the appropriate way of applying out MBT tool is divided into a set of activities. Roughly divided, each activity can be associated with one model transformation (pink ovals in component diagram).

We propose the following steps to be taken when applying the MBT tool:

1) ATL transformation (*generate Transition Tree from State Machine*)
2) Generate test data (*only first time of use, once for each unique domain model*)
3) MofScript transformation (*insert test-data/grammar, generate test-cases*)

The UML Activity Diagram below explicitly shows how the model-based testing automation is achieved by performing activities defined in the context of the MBT tool.

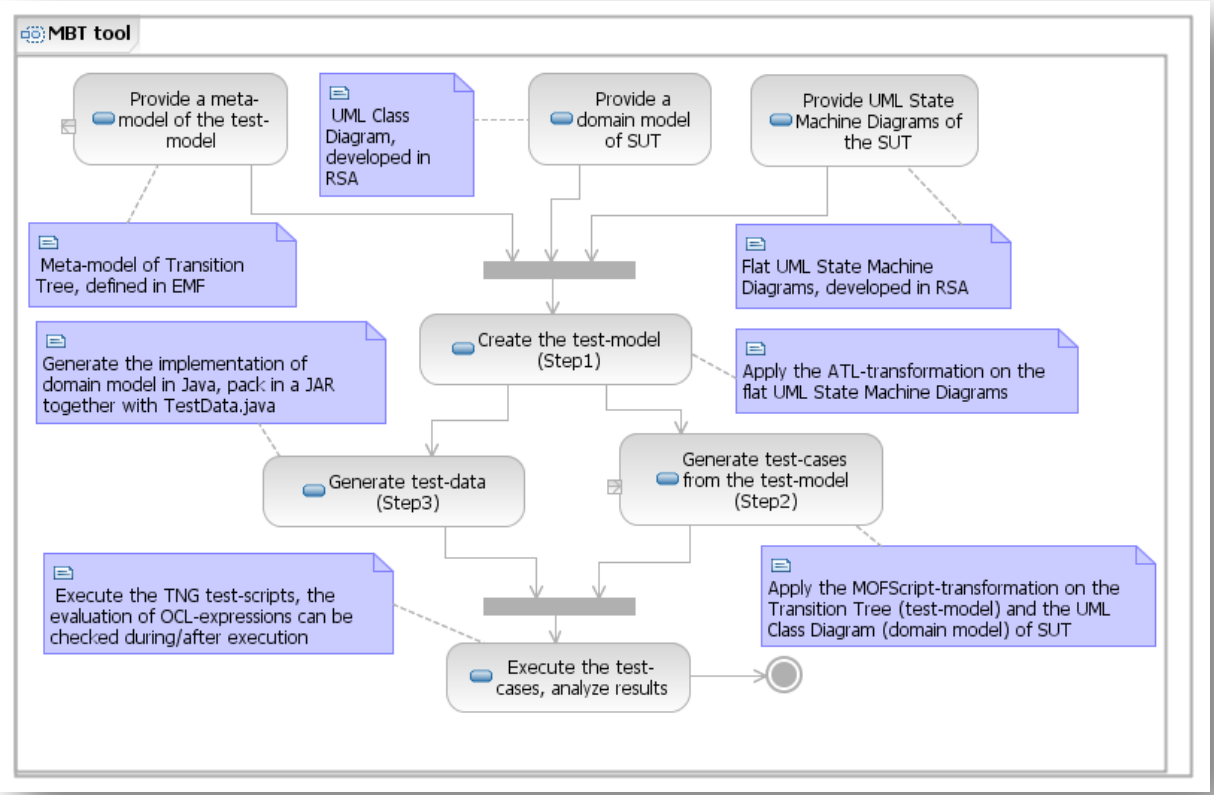


Figure 16 – Development activities of the MBT tool

Prior to performing activities illustrated in Figure 16, one must have the appropriate plugins installed. These are the EMF, ATL and MOFScript plugins. The UML meta-model is usually present in all Eclipse versions, so this model is present in RSA as well.

The generation of the implementation of Class- and Object diagrams in Java must be done for each unique UML Class Diagram. If we later want to produce test-cases associated with the same UML Class Diagram, we can just reuse the previously generated file (*ClassDiagramTestData.java*).

The generated test-cases (TNG scripts), can be put in the TNG folder (in C:\), together with the Get_Val component (Section 4.4.4), IntGenerator component (Section 4.4.5), Test component (Section 4.4.1 + Section 4.4.3), the setup file (Section 4.4.6) and the IEOS component (Section 4.4.2). Having these settings in order, the scripts can now be executed, and the results collected.

6 Relevant technical details

In this chapter I will explain about my contribution to the project. Section 6.1 presents the first approach we tried out, without success. The remaining sections explain my contribution to the developed MBT tool.

6.1 A different approach

We have tried two different approaches in development of our MBT tool. The largest amount of time and effort was spent on the approach described in this thesis, involving model transformations, and the challenges involved in this process. In this section I will describe the first approach that we wanted to use.

In the start of my thesis, autumn 2009, we had a completely different approach we wanted to use to develop the MBT tool. This approach utilized models as well, but it did not involve model transformations. I will here only briefly describe this approach for documentation and also as help to others that may follow the same path.

The idea behind this approach was to use Eclipse, [22], in combination with Papyrus,[23], in order to develop the models. The models used in this approach were UML *Sequence Diagrams*. The objectives here were to use the diagram's XML representation and parse it using a parser implemented in Java. The parser would translate diagram's XML representation to a specific data structure called *Interprocedural Restricted Control-Flow Graph*, IRCFG. This process is depicted in Figure 17.

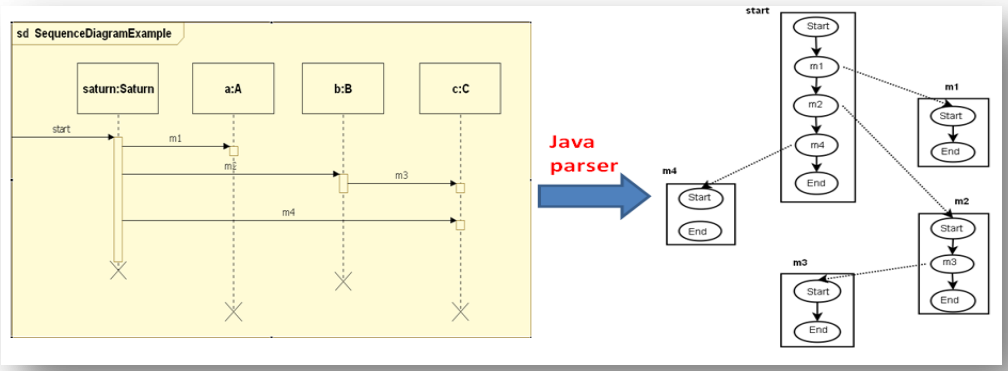


Figure 17 – Translating a UML 2.0 Sequence Diagram into IRCFG

The example above illustrates a very simple example, where the modeled method-calls are not associated with any constraints. In practice this scenario is rare, but I have nevertheless made this small example to illustrate the basic process of deriving an IRCFG from a Sequence Diagram modeled in Papyrus. A more comprehensive case study, without implementation details, can be found in [24].

The derived IRCFG is an abstract model of a part of the system under test. Of course, if the system or program is relatively small, it would be possible to map all of the interactions taking place in a single Sequence Diagram. But for complex software systems this is often not possible to achieve.

The IRCFG derived would then make it possible to define a formal definition of the *coverage criteria* for the system (or code) under test. For instance, if the desired coverage criteria were the *all-paths criterion*, then we would achieve this by covering the entire set of complete IRCFG paths. In Figure 17 we can see that there is just one complete path, as we do not have any pre-conditions on the object interactions.

The structure of IRCFG can also be used for runtime analysis. I will not go into further details about this approach here, as we were only in the starting point when we decided to abandon it. The code, parser, is added as Appendix A – SD2IRCFG.java to this work.

This approach did not work very well for us. A number of problems were identified during the implementation of the proposed solution. Briefly, there were technical difficulties with the tool (Papyrus), for which we used a lot of time to solve. These concerned the sequences of the methods specified in the Sequence Diagram.

Also, the user had to be very careful while drawing the diagram, as it was easy to make errors. For example, if a user pointed the tip of the arrow representing a method-call (create/reply/synchronous/asynchronous) any other spot other than the left upper corner of the targeted *Behavior Execution*, the XML schema of the diagram was incorrect. In simpler terms, one could not derive what was specified inside the body of a method-call. This made very difficult to efficiently derive the IRCFG for a UML 2.0 Sequence Diagram.

The approach had other flaws as well. As it operates on model-level and not the meta-model level, the XML-representation of the Sequence Diagram from previous version of Papyrus may be different from the current version. This means that a parser implemented for one version of Papyrus may not work for other versions. We used Papyrus version 1.10 and the implemented parser/translator may not work on later versions of Papyrus, as the naming of elements in the XML-schema may differ. Nevertheless, the code is added to this thesis both to document my own work on this particular solution proposal and to give interested readers the idea of the algorithm used to derive the IRCFG.

6.2 My contribution to the project

My contributions to the project are twofold. Firstly I have written the *translator* presented in the previous section (Section 6.1), in the context of the solution we did not proceed with. Secondly, I have contributed to the development of the successfully developed MBT tool. Even though the objectives of the two solutions are the same (automated generation of test-cases derived from system specifications), the two approaches are very different. The following sections present contributions regarding the MBT tool.

6.2.1 Input models

There are four models required by the MBT tool, and three of these we needed to develop ourselves, the UML meta-model was already developed and present in RSA. The other three models are the Transition Tree meta-model, the UML State Machine Diagram and the UML Class Diagram (domain model). The initial versions of the Transition Tree meta-model and the domain model were developed by my co-students. During development of the tool we saw the need for change several places. These changes were often based on a lack of detail in the meta-model or we needed to modify the structure of the (meta-) model itself. I have in this context contributed to the evolution of the Transition Tree meta-model and the domain model used by the developed MBT tool.

6.2.2 Model transformations

There are three model transformations currently implemented in the MBT tool. These are the two model transformations that generate the test-cases and the model transformation that generates the Class and Object diagrams in Java (Section 4.4.3). I have written these model transformations following the strategy set by my supervisor and my co-students.

6.2.3 Test-data generation

I have contributed to the test-data generation in several ways. I have participated in development of the Get_Val component (Section 4.4.4), and I have written the TestData component (Section 4.4.5). I have also written the model transformation that generates the Class- and Object diagram in Java presented above.

6.2.4 Evaluation

I have done my own evaluation of the MBT tool separately from the more thorough evaluation that will be done by my supervisor and my co-students. I have paid most attention at the correctness (both semantic and syntactic) of the generated test-cases and the evaluation of the constraints that are defined. The SUT also has a timer so we are able to measure the time needed to execute a test-case. This information can be further used for comparisons. Test-cases developed manually, without any Python sections and external method-calls, containing pure TNG code, are likely to finish their execution sooner than the test-cases derived automatically. This is because the test-cases produced by the tool are bigger and contain more code to be executed. Below we is a screenshot showing the results of executing an automatically derived test-case produced by the MBT tool.

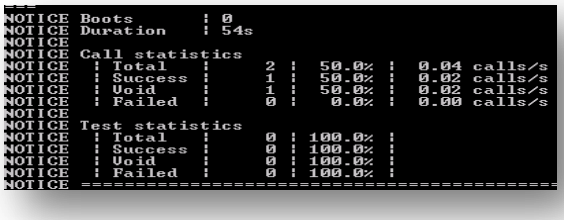


Figure 18 - Result of test-case execution

As we can see in the screenshot above, the generated test-cases are syntactically correct, and there are no failed method invocations (calls). The table shows other test characteristics as well. The screenshot in Figure 19 shows another screenshot showing the collected results of the evaluation of the OCL expressions.

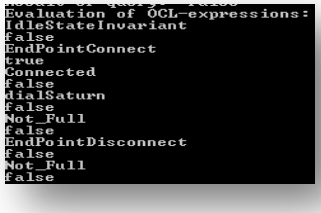


Figure 19 - Results of evaluating the specified constraints

As we can derive from the above screenshot, the constraints are evaluated at runtime and the results are collected and displayed at the end of execution. All the test-cases that were executed were running on machines at Tandberg, which are frequently used by others as well. As a result of this, we could not fully control the states the SUT was in, because of those external factors (such as someone calling the test-target, SUT, while a test-case is executing). The example implies the construction of test-beds, isolated environment, when running the test-cases, in order to get the most precise results. The test-case generating the output presented in the above figures can be inspected in Appendix M– Sample test-case.

6.2.5 Example of use

In this section I will illustrate how to apply the MBT tool in order to automatically generate executable test-cases. The example follows the three steps proposed in Chapter 5.

6.2.5.1 ATL transformation (generate Transition Tree from State Machine)

In this first step of the transformation process we are interested in deriving a Transition Tree from UML State Machine Diagram. The ATL-module used can be inspected in Appendix E – StateMachine2TransitionTree.atl (ATL). The input models to this transformation are the source model, shown in Figure 20, and the two meta-models, UML meta-model and Transition Tree meta-model. The two meta-models can be checked in Appendix B – UML 2.0 Meta-Model (UML) and Appendix C – Transition Tree Meta-Model (EMF). In addition to these models, we must specify a name for the generated target model in ATL Run-Configurations (*atlOutput.xml* in our case). The figure below shows the used UMLState Machine Diagram (RSA).

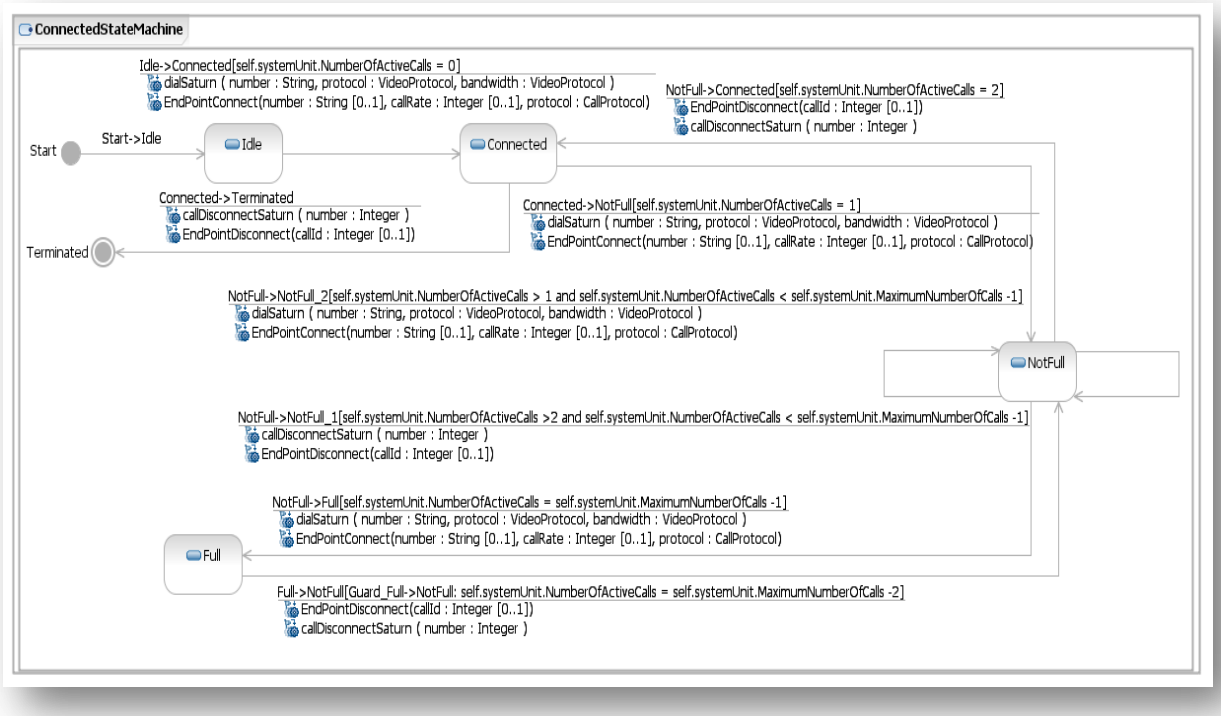


Figure 20 - UML State Machine Diagram

The UML State Machine Diagram serves as basis for further test-case generation. The states from this State Machine are mapped to nodes, constraints are mapped to constraints, outgoing transitions are mapped to transitions, and so on. The complete Transition Tree is too big to present in this illustration, so I will only insert here the section that shows the structure of the generated Transition Tree. The code can be checked on the following page.

```
?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:transitiontree="http://transitiontree/1.0">
  <transitiontree:TransitionTree>
    <alpha name="Start">
      <transitions name="Start->Idle">
        <children name="Idle->Connected" guard="/5" trigger="/55 /70">
          <children name="Connected->NotFull" guard="/1" trigger="/58 /73">
            <children name="NotFull->Connected" guard="/11" trigger="/74 /65">
              <target name="Connected" stateInvariant="/6"/>
            </children>
            <children name="NotFull->Full" guard="/14" trigger="/67 /76">
              <children name="Full->NotFull" guard="/2" trigger="/77 /64">
                <target name="NotFull" stateInvariant="/15"/>
              </children>
              <target name="Full" stateInvariant="/9"/>
            </children>
            <children name="NotFull->NotFull_1" guard="/4" trigger="/66 /68">
              <target name="NotFull" stateInvariant="/15"/>
            </children>
            <children name="NotFull->NotFull_2" guard="/13" trigger="/63 /69">
              <target name="NotFull" stateInvariant="/15"/>
            </children>
            <target name="NotFull" stateInvariant="/15"/>
          </children>
          <children name="Connected->Terminated" trigger="/62 /75">
            <target name="Terminated" isTerminal="true"/>
          </children>
          <target name="Connected" stateInvariant="/6"/>
        </children>
        <target name="Idle" stateInvariant="/8"/>
      </transitions>
    </alpha>
  </transitiontree:TransitionTree>
```

The code snippet above shows the structure of the Transition Tree (test-model), the whole file can be checked in Appendix F - The generated Transition Tree (XML). After executing the ATL transformation we can proceed with the next step of the transformation process.

6.2.5.2 Generate test data (only first time of use, once for each unique domain model)

As the title suggests, this step is applied in order to generate the implementation of Class and Object diagrams in Java (Appendix I – ClassDiagramTestData.java (Java)), derived from the domain model of Saturn (Appendix D – The domain model of Saturn (UML)), using MOFScript (Appendix H – ClassDiagramTestdata.m2t (MOFScript)). The generated file is packed, together with the TestData.java component, in a Jar file and put in the TNG directory, from where it can be used during runtime.

This step can be omitted if earlier undertaken, and no changes to the domain model have been added. This step has no impact on the generation of test-cases themselves, but is crucial in order for us to be able to evaluate the specified constraints (OCL expressions).

6.2.5.3 MofScript transformation (insert test-data/grammar, generate test cases)

The last step of the transformation process generates the executable test-cases, taken as input the Transition Tree from the first step of the transformation process and the UML Class Diagram (domain model). As previously mentioned, each generated test-case covers one path in the Transition Tree. The model transformation used can be inspected in Appendix G – SM2TT.m2t (MOFScript). The generated test-cases are too big to present in this example, so I have only inserted the section that checks the *Connected* state of Saturn (Figure 20) in the code snippet below.

```
……
# State: Connected
# StateInvariant:Connected    Value: self.systemUnit.NumberOfActiveCalls=1 and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off or
c.outgoingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off)->size() =0
eval = 'Connected'
# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
        expression = attRole[index]+'.'+name
        if int(attMult[index]) > 1:
                value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
        else:
                value = Get_Val.new_get_value(target,attPath[index])
        if len(value)-1 == 0:
                j = 0
        if expression in "self.systemUnit.NumberOfActiveCalls=1 and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size() =0  ":
                existAtt += " " + attClass[index]
                existAtt += " " + name
                existAtt += " " + attObject[index]
                existAtt += " " + attType[index]
                theValue = " '"  + value[j] + "'"
                if attType[index] == 'Integer':
                        theValue = " " + value[j]
                existAtt += theValue
        if j < (len(value)-1):
                j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls=1 and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size() =0")
print "Result of query: ", queryResult
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
……
```

The code snippet above illustrates how the state invariant of the *Connected* state is evaluated. Similar code segments are generated when dealing with guards on transitions. The model transformation generates five test-cases, each corresponding to one path from the Transition Tree (Figure 6). The test-cases are put in the TNG directory and can executed from there. A complete sample test-case can be inspected in Appendix M– Sample test-case, for more details.

6.3 Problem examples and Lessons Learned

This section discusses the most important obstacles we had to overcome in order to develop our MBT tool successfully. I have divided these obstacles into three different classes, namely problems related to *models, test-case generation* and *test-data generation*. The following sections describe each of these in turn, and propose a possible solution.

6.3.1 Problems related to models

As presented earlier, there are two types of diagrams needed as input to the MBT tool. These are the UML State Machine Diagram, and the associated UML Class Diagram where the triggers (API calls and Signal Receptions), effects and other elements from the UML State Machine Diagram are specified.

Now, having these models in mind, it is vital that the domain model (UML Class Diagram) be a correct representation of the SUT. This might seem as an easy challenge, but in a highly dynamic environment, where changes are made constantly, keeping the domain model consistent can be a problem. Through conversation with an employee at Tandberg I was informed that changes to e.g. system variable names and types do occur, as well as other changes to the implementation of the SUT. Also, the change is not announced, so the testers mostly find out about these changes during testing.

Solving this problem would require having a domain model which is always updated. Making a change in the implementation of the system would require first a corresponding change in the domain model.

Another problem with models can occur if the models are not specified correctly or missing some relevant information. The information specified in the domain model and the state machine is of course crucial for the generation of test-cases. This information must be specified fully and correctly. Omitting one important detail, e.g. not specifying the type of a parameter, will result in the test-cases being buggy.

Keeping the two models complete is essential for the tool to work properly. We can avoid these kinds of problems by updating the models consistently and in the manner specified in the original version of the models that we used.

The current version of the tool is packaged as an ATL project. We have experienced some smaller problems when importing the tool to another workspace. When importing the tool, the user must make sure the *whole* package is imported. Most important is it that the diagrams are imported correctly.

6.3.2 Problems related to test-case generation

The biggest challenge for me was the way one had to think while writing the model transformations. I have, from previously attended courses, experience with several programming languages, but writing model transformation is very different. Both model-to-model transformations (m-2-m) and model-to-text (m-2-t) transformations are on a high abstraction level, so the programmer has to get used to think accordingly. Both of these transformation types are powerful tools in MDD. As earlier mentioned, we have used ATL for m-2-m transformations and MOFScript for m-2-t transformations.

The biggest problem writing the m-2-m transformation (ATL-transformation) was to find a way to express what we wanted the model transformation to produce, using the constructs available in ATL. I had little knowledge about OCL earlier, so this was a challenge since the preferred programming style of ATL is declarative. ATL provides imperative constructs as well. These are needed in order to ease the specifications of mappings that are hard to do declaratively, e.g. using recursive rules.

The m-2-t transformations, using MOFScript, I found much easier to comprehend and more natural to understand. The objective of these transformations is to produce syntactically correct code segments and files. The programming constructs in MOFScript are similar to that of Java, so in my own opinion, it was much easier to write these transformations compared to the ATL transformation.

The best way of tackling these programming challenges is by practice and using the available documentation regarding ATL ([7]) and MOFScript ([14]). Running the ATL transformations in *debug* mode will provide some support, but I still found it very difficult to locate bugs even in debug mode. Also, a good insight into MDA ([3], [5]) is important in understanding the way model transformations must be specified to produce a set of target model elements, or code segments, from a set of source model elements.

Another challenge regarding the generation of test-cases was to make all of the components, which constitute the MBT tool, work together correctly. The tool is made out of many parts, implemented in different programming languages, so coordination is important.

An incremental approach was used to develop the components of the MBT tool. This made it easier to tackle the complexities introduced by the number of different components. It was also easier for me as a programmer since I could concentrate at one component at a time.

The problems and challenges presented in this sub-chapter are easy to overcome once one has gotten a little experience and practice using model transformations. As mentioned earlier, when one has gotten used to think in *models*, the model transformations become important and powerful tools for working with models.

6.3.3 Problems related to test-data generation

The biggest challenge regarding the test-data generation was to implement the mechanisms that check the defined constraints (e.g. guards and state invariants) at runtime. As previously mentioned, this is done by having a partly consistent domain model in Java (Section 4.4.3), and querying this model during the execution of the test-cases. The list below describes some of the most relevant problems in this context.

(1) The Class and Object diagram in Java (Section 4.4.3) might not have been derived correctly, causing Java exceptions thrown from inside the OCL evaluator (Section 4.4.2) when a test-script is executed.

(2) The OCL expressions (constraints) in the UML State Machine Diagram can be misspelled, also causing exceptions to be thrown from inside the OCL evaluator when the test-script is executed.

(3) The OCL evaluator can also throw exceptions if we try to update the object model with inappropriate values (or types) of the object attributes (Section 4.4.4).

(4) The TestData.java component will return invalid IP addresses of endpoints if the setup file is not up to date.

The first two problem examples are solved by careful inspection of the two input models, UML Class Diagram (domain model) and UML State Machine Diagram. A systematic approach to updating and modifying the models is recommended. Each detail in the models is important, and must be specified with care. The syntax of the predicates (constraints) must be checked and verified.

The Get_Val module will return the current value of a system variable, based on the path and sub-path specified in the domain model. This path and sub-path must be correct (relative to the SUT), since the returned value(s) will be used to update the Object Model. JPype (Section 3.3.5) must also be installed correctly, since this makes it possible to update, query and get results from the OCL evaluator.

The last problem example is the least serious one. The problem occurs if we try to probe an endpoint that is no longer there, e.g. because the IP addresses are incorrect or the file itself is outdated. These exceptions are handled by the system, and it is easy to identify the erroneous endpoint(s). To overcome situations such as these, the tester is encouraged to verify the setup file prior to test execution.

7 Conclusion

7.1 Contribution of my thesis

The contribution of my thesis is a prototype MBT tool that can be used for automated Model Based Testing of Tandberg's C90 codec, Saturn. The tool presents a solution to a Model Based Testing approach, based on behavioral models (UML State Machines). The tool was developed with the intention to automate the testing processes undertaken at the test-department of Tandberg. The tool uses models and model transformations, a MDD approach, to acheve its objectives. Even though the tool is at its earliest stages, and there are still a lot of things to improve, it nevertheless provides the testers with a practical and efficient technique for model-based testing of Saturn. The tool is much more efficient in generating the test-cases than the manual technique, and can be used in combination with other testing techniques in order to achieve a greater confidence in the correctness of the final product.

7.2 Future work and limitations of the current tool

There are a number of limitations associated with the current version of the MBT tool presented in this thesis. Most important is that the current version of the tool can only be applied to model-based testing of Tandbergs C90 codec, Saturn. For example, even if the setup file (*test.tts)* contains IP addresses of other types of codecs (C60,C30), we may get void invocations, since the other codecs may not support the picked bandwidth or/and call protocol. Therefore, in order to avoid such situations, we should only involve Saturn machines in the test-environment of the SUT, if possible.Another limitation is that the tool can only handle *flat* (non-hierarchical) UML State Machine diagrams developed in RSA. Small changes to the ATL and MOFScript transformations are necessary if the tool is to be exported to other platforms (tools), such as Eclipse for example, because of the specific packaging format (*.emx*) of the RSA. The same conclusion can be drawn for the domain model (UML Class Diagram).

As the tool is based on model transformations and consisits of many parts, each presents potential improvements (or changes) regarding the future work. The most important component is the Transition Tree (test model), produced by the ATL transformation. It would be worth exploring if we can produce a similar test-model from other types of diagrams (e.g.*Sequence/Activity Diagrams*). We need only to write a different ATL module in order to achieve this. The same procedure can be applied when we want to generate test-cases with other syntax and/or grammar. Here we need to write a suiting MOFScript transformation, that will provide the correct syntax and grammar of the target system. Using the described approach, we can in theory test any system who's behavior can be expressed using a behavior model, given that we have provided model transformations that can handle the type of model.

There are a number of improvements related to test-data components to be considered as well. A more systematic approach to generation of parameter values can be applied, in contrast to randomly picking the values from a pool of possible values, as is the case in the current version of the tool. A possibility to generate various sequences of API calls, with the intention of focusing specific class of faults, can be implemented as well. The tool can then be evaluated in terms of *Stress* and *Robustness Testing* techniques. The tool can also be adjusted to cover different coverage criteria.

The mechanism for verifying the specified constraints could also be improved. The mechanism now uses one external call every time one OCL expression is evaluated, making this an expensive solution if the number of expressions to be evaluated is large. An equivalent, automated, solution implemented in Python would be ideal, also because the testers at Tandeberg have explained that they are now in the process of migrating over to Python completely. The expressions could then be evaluated locally in the test-case, making the tool even more efficient, practical and less time-consuming.

8 Reference list

1. *JUnit*. [Internet](25.05.2009); Available from: http://www.junit.org/.
2. *TANDBERG Home*. [Internet] (24.03.2009); Available from: http://*www.tandberg.no*
3. *IBM, MDA Documentation*. [Internet] (15.04.2009); Available from: http://www.ibm.com/developerworks/rational/library/3100.html.
4. Ambler, S.W., *The Elements of UML 2.0 Style*. 2005: Cambridge University Press.
5. *OMG, MDA Documentation*. [Internet] (14.04.2009); Available from: http://www.omg.org/mda/.
6. Pezze, M. and M. Young, *Software Testing and Analysis*. 2008: John Wiley & Sons, Inc.
7. ATLAS group, L.I. (2006) *ATL User Manual*. **0.7**.
8. K.Nagin, A.H.a., *The AGEDIS Tools for Model Based Testing.* 2004.
9. Brinksma, J.T.a.E., *TorX:Automated Model Based Testing.* 2002.
10. Faivre, J.D.a.A., *Automating the Generation and Sequencing of Test Cases from Model-Based Specifications.*
11. A.Pretschner, W.P., M.Baumgartner,T.Stauner, *One Evaluation of Model-Based Testing and its Automation.* 2005.
12. *Rational Software Architect*. [Internet] (27.05.2009); Available from: http://www-01.ibm.com/software/awdtools/architect/swarchitect/.
13. *Eclipse Modeling Framework*. [Internet] 2009 28.05.2009]; Available from: http://www.eclipse.org/modeling/emf/.
14. J.Oldevik (2005) *MOFScript User Guide*. **0.1**.
15. *MOFScript Home*. [Inrenet] (23.03.2009); Available from: http://www.eclipse.org/gmt/mofscript/.
16. *JPype Home*. [Internet] (05.05.2009); JPype HomePage]. Available from: http://jpype.sourceforge.net/.
17. *UML Resources*. [Internet] 20.03.2009]; Available from: *http://www.uml.org/*
18. Pilone, D. and N. Pitman, *UML 2.0 in a Nutshell*. 2005: O'Reilly Media,Inc.
19. *Finite State Machines*. [Internet] [cited 2009 30.03.2009]; Available from: http://en.wikipedia.org/wiki/Finite_state_machines.
21. Binder, R.V., *Testing Object-Oriented Systems: Models, Patterns, and Tools*. 1999: Addison-Wesley,p.261.
22. *Eclipse Home*. [Internet] (26.03.2009); Available from: http://www.eclipse.org/.
23. *Papyrus Home*. [Internet] (12.10.2008); Available from: http://www.papyrusuml.org/.
24. Rountev, A., S. Kagan, and J. Sawin, *Coverage Criteria for Testing of Object Interactions in Sequence Diagrams*. 2005.

9 Appendixes

9.1 Appendix A – SD2IRCFG.java

```
import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;

/*
 *@author Miran Damjanovic
 *@date 19.10.2008
 *The class SD2IRCFG is the main class in this translator/parser.
 *It owns all the implemented functions in this temporary parser.
 *Many of the details are currently not looked into, the main
 *purpose of this parser is to check weather using this approach
 *is efficient and possible in automatically creating test-cases.
 *@param xmlFileName String the XML-file to parse
 *@param v Vector holds the Nodes-objects
 *@param w Vector holds the Fragment-objects
 *@param n Nodes pointer to objects of type Nodes
 *@param f Fragment pointer to objects of type Fragment
 *@param global int global counter for sequencing Nodes and Fragments
 */
public class SD2IRCFG {
	private final static String xmlFileName = "test.xml";
	public static Vector v = new Vector();
	public static Vector w = new Vector();
	public static Nodes n;
	public static Fragment f;
	int global = 0;

	/*
	 * The main function, initialize new SD2IRCFG object, which calls the
	 * constructor. The constructor starts the translation process by invoking
	 * the other functions in turn.
	 */
	public static void main(String[] args) {
		new SD2IRCFG();
	}

	/*
	 * The constructor for class SD2IRCFG. invokes the functions that handle the
	 * xml-file and produce the IRCFG structure.
	 * @param doc Document the XML-document
	 * @param root Node root of the XML-structure
	 */
	public SD2IRCFG() {
		Document doc = parseFile(xmlFileName);
		Node root = doc.getDocumentElement();
		// extract data from XML-tree
		assignProperties(root, 0);
		deriveProperties();
		derivePropertiesForFragments();
		orderCoveredBy();
		orderInvokations();
```

```
		checkForExternalCall();
		derivePropertiesBehExec();
		deriveRCFG();
		deriveRCFGEdges();

	}

	/*
	 * Function returns values of child nodes for a given element defined in the
	 * XML_file.
	 * @param elem Node the parent node
	 * @param child Node the child node
	 * @return the child node (as String) or empty String
	 */
	public final static String getElementValue(Node elem) {
		Node child;
		if (elem != null) {
			if (elem.hasChildNodes()) {
				for (child = elem.getFirstChild(); child != null; child = child
						.getNextSibling()) {
					if (child.getNodeType() == Node.TEXT_NODE) {
						return child.getNodeValue();
					}
				}
			}
		}
	return "";
	}

	/*
	 * This fuction initializes all the Nodes objects and the Fragment objects,
	 * and derives their attribute values.
	 * @param index int counter
	 * @param node Node a Node object
	 * @param nodeName String name of the node
	 * @param attributes NamedNodeMap attributes owned bu node
	 * @param children NodeList a NodeList object
	 * @param n Nodes a Nodes object
	 * @param f Fragment a Fragment object
	 */
	public void assignProperties(Node node, int index) {
		String nodeName = node.getNodeName();
		NamedNodeMap attributes = node.getAttributes();
		n = new Nodes();
		for (int i = 0; i < attributes.getLength(); i++) {
			Node attribute = attributes.item(i);
			if (nodeName.equals("lifeline")) {
				n.type = "Class";
				if (attribute.getNodeName().equals("coveredBy")) {
					n.lifelines = orderLifelines(attribute.getNodeValue());
				}
				if (attribute.getNodeName().equals("name")) {
					n.name = attribute.getNodeValue();
				}
				if (attribute.getNodeName().equals("xmi:id")) {
					n.id = attribute.getNodeValue();
				}
				if (i == attributes.getLength() - 1) {
					v.add(n);
				}
```

```
		}
	}

	for (int i = 0; i < attributes.getLength(); i++) {
		Node attribute = attributes.item(i);
		if (nodeName.equals("message")) {
			n.type = "Message";
			if (attribute.getNodeName().equals("receiveEvent")) {
				n.receiveClass = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("sendEvent")) {
				n.sendClass = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("xmi:id")) {
				n.id = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("name")) {
				n.name = attribute.getNodeValue();
			}
			if (i == attributes.getLength() - 1) {
				v.add(n);
			}
		}
	}
	f = new Fragment();
	for (int i = 0; i < attributes.getLength(); i++) {
		Node attribute = attributes.item(i);
		if (nodeName.equals("fragment")) {
			if (attribute.getNodeName().equals("xmi:type")) {
				f.type = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("covered")) {
				f.covered = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("name")) {
				f.name = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("message")) {
				f.message = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("event")) {
				f.event = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("execution")) {
				f.execution = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("finish")) {
				f.finish = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("start")) {
				f.start = attribute.getNodeValue();
			}
			if (attribute.getNodeName().equals("xmi:id")) {
				f.id = attribute.getNodeValue();
			}
			if (i == attributes.getLength() - 1) {
				w.add(f);
			}
		}
	}
```

```
		NodeList children = node.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);

			if (child.getNodeType() == Node.ELEMENT_NODE) {
				assignProperties(child, index + 2);
			}

		}
	}

	/*
	 * The function opens the XML-file, and parses it using the Java
	 * DocumentBuilder class.
	 * @param fileName String the name of the XML-file
	 * @param docBuilder DocumentBuilder a DocumentBuilder object
	 * @param doc Document a Document object
	 * @param docBuilderFactory DocumentBuilderFactory a DocumentBuilderFactory
	 * object
	 * @param souceFile File a File object
	 * @return the produced document (doc)
	 */
	public Document parseFile(String fileName) {
		System.out.println("Starting parser.." + fileName);
		DocumentBuilder docBuilder;
		Document doc = null;
		DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
				.newInstance();
		docBuilderFactory.setIgnoringElementContentWhitespace(true);
		try {
			docBuilder = docBuilderFactory.newDocumentBuilder();
		} catch (ParserConfigurationException e) {
			System.out.println("Error: " + e.getMessage());
			return null;
		}
		File sourceFile = new File(fileName);
		try {
			doc = docBuilder.parse(sourceFile);
		} catch (SAXException e) {
			System.out.println("Error in XML-file: " + e.getMessage());
			return null;
		} catch (IOException e) {
			System.out.println("File Error: " + e.getMessage());
		}
		return doc;
	}
```

```
/*
 * This functions iteratesover all Fragments (held in Vector w) and derives
 * relevant attributes for each Behavior Execution.
 * @param f2 Framgment temporary object of type Fragment
 * @param f3 Framgment temporary object of type Fragment
 */
public void derivePropertiesBehExec() {
        for (int i = 0; i < w.size(); i++) {
                f = (Fragment) w.elementAt(i);
                if (f.name.startsWith("ExecSpec_")) {
                        for (int j = 0; j < v.size(); j++) {
                                n = (Nodes) v.elementAt(j);
                                if (n.type.equals("Class")) {
                                        for (int k = 0; k < n.coveredBy.size(); k++) {
                                                Fragment f2 = (Fragment)
                                                n.coveredBy.elementAt(k);
                                                if (f2.name.startsWith("StartEO_")
                                                        && f2.execution.equals(f.name)) {
                                                        Fragment f3 = null;
                                                        if (k == 0) {
                                                        f3 = (Fragment) n.coveredBy
                                                                        .elementAt(k+ 1);
                                                        }
                                                        if (k > 0) {
                                                        f3 = (Fragment) n.coveredBy
                                                                        .elementAt(k - 1);
                                                        }

                                                        if (f3.name.startsWith("Receive")) {
                                                                f.method = f3.message;
                                                        }
                                                        k += 1;
                                                f3=(Fragment)n.coveredBy.elementAt(k);
                                                while(!(f3.name.startsWith("FinishEO"))
                                                                        &&
                                                        !(f3.name.startsWith("Receive"))) {
                                                                f.mBody.add(f3.message);
                                                                k++;
                                                f3 = (Fragment) n.coveredBy.elementAt(k);
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }

}
```

```
/*
 * This function derives the RCFG structure (held in a Vector) for each
 * Nodes object. After executing this function, every message object has an
 * Vector holding the methods/functions that can be invoked from its
 * message-body.It also adds the Start and End Nodes to the RCFG Vector.
 *
 * @param start Nodes the start node of a RCFG
 *
 * @param end Nodes the end node of a RCFG
 */
public void deriveRCFG() {
	for (int i = 0; i < v.size(); i++) {
		n = (Nodes) v.elementAt(i);
		if (n.type.equals("Message")) {
			for (int j = 0; j < w.size(); j++) {
				f = (Fragment) w.elementAt(j);
				if (f.method != null) {
					if (f.method.equals(n.name)) {
						Nodes start = new Nodes();
						start.name = "Start";
						if (!containsByName(n.rcfg, start)) {
							n.rcfg.add(start);
						}
						if (f.mBody != null) {
							for (int k = 0; k < f.mBody.size(); k++) {
								String m = (String)
								f.mBody.elementAt(k);
								for (int l = 0; l < v.size(); ++) {
								Nodes n2 = (Nodes)
								v.elementAt(l);
								if (n2.name.equals(m)
										&&
								!containsByName(n.rcfg, n2)) {
								n.rcfg.add(n2);
										}
								}
							}
						}
						Nodes end = new Nodes();
						end.name = "End";
						if (!containsByName(n.rcfg, end)) {
							n.rcfg.add(end);
						}
					}
				}
			}
		}
	}
}
```

```
/*
 * Helper function that takes a Vector and Nodes object and checks if node
 * is contained in the given Vector, the check is performed on the name of
 * the Nodes object.
 *
 * @param contains Boolean true if node is present in vector,false otherwise
 *
 * @param n3 Nodes a tamporary object of type Nodes return true if found
 * name of node in vector, false otherwise
 */
public Boolean containsByName(Vector h, Nodes n2) {
	Boolean contains = false;
	if (h != null) {
		for (int i = 0; i < h.size(); i++) {
			Nodes n3 = (Nodes) h.elementAt(i);
			if (n3.name.equals(n2.name)) {
				contains = true;
			}
		}
	}
	return contains;
}

/*
 * This function adds the fragments to the coveredBy Vector belonging to a
 * node (type Nodes), according to the id's of fragments found in lifelines
 * Vector of the node.
 */
public void orderCoveredBy() {
	for (int i = 0; i < v.size(); i++) {
		n = (Nodes) v.elementAt(i);
		if (n.type.equals("Class")) {
			for (int j = 0; j < n.lifelines.size(); j++) {
				for (int k = 0; k < w.size(); k++) {
					f = (Fragment) w.elementAt(k);
					if (f.id.equals(n.lifelines.elementAt(j))) {
						n.coveredBy.add(f);

					}
				}
			}

		}
	}
}
```

```
/*
 * This method parses the string containing the lifeline id's for a class
 * from Sequence Diagram, and returns the result as a vector. In short,it
 * transforms a String value to a Vector containing the fragment id's.
 * @param l String the String containing the id's of fragments in the
 * lifeline
 * @param vl Vector the Vector to be returned
 * @param st StringTokenizer an obect of type StringTokenizer
 * @return the Vector,vl,containing the fragment id'd
 */
public Vector orderLifelines(String l) {
        Vector vl = new Vector();
        StringTokenizer st = new StringTokenizer(l);
        while (st.hasMoreTokens()) {
                vl.add(st.nextToken());
        }
        return vl;
}

/*
 * This method assigns values to the two vectors getsInvocation and invokes
 * owned by every node.The getsInvokation Vector more precisely contains the
 * methods that are implemented in the class (from Sequence Diagram)
 * corresponding to the given node. The invokes Vector contains the methods
 * that can be invoken from the class (from Sequence Diagram) corresponding
 * to the given node.
 *
 * @param n2 Nodes a temporary object of type Nodes
 */
public void orderInvokations() {
        if (v != null) {
                for (int i = 0; i < v.size(); i++) {
                        n = (Nodes) v.elementAt(i);
                        if (n.type.equals("Class")) {
                                for (int j = 0; j < v.size(); j++) {
                                        Nodes n2 = (Nodes) v.elementAt(j);
                                        if (n2.type.equals("Message")) {
                                                if (n2.receiveClass.equals(n.name)) {
                                                        n.getsInvokation.add(n2);
                                                } else if (n2.sendClass.equals(n.name)) {
                                                        n.invokes.add(n2);

                                                }
                                        }
                                }
                        }
                }
        }
}
```

```
/*
 * This method simply finds the values for sending and receiving Objects of
 * every function from the Sequence Diagram, and at the same time looks for
 * preconditions and setd those if there are any present in diagram.
 * @param s StringBuffer an object of type StringBuffer
 * @param s2 StringBuffer an object of type StringBuffer
 * @param t String a temporary String
 * @param r int a temporary variable used to hold index of a character
 */
public static void deriveProperties() {
        StringBuffer s, s2;
        int r;
        if (v != null) {
                for (int i = 0; i < v.size(); i++) {
                        n = (Nodes) v.elementAt(i);
                        if (n.type.equals("Message")) {
                                for (int j = 0; j < v.size(); j++) {
                                        Nodes n2 = (Nodes) v.elementAt(j);
                                        if (n2.type.equals("Class")) {
                                                if (n2.lifelines.contains(n.sendClass)) {
                                                        n.sendClass = n2.name;
                                                        } else if
                                                        (n2.lifelines.contains(n.receiveClass)) {
                                                        n.receiveClass = n2.name;
                                                }
                                        }
                                }
                                if (n.name.contains("]") && n.name.contains("[")) {
                                        String t = n.name.trim();
                                        s = new StringBuffer();
                                        s2 = new StringBuffer();
                                        if (t.startsWith("[")) {
                                                char c;
                                                r = t.indexOf("]");
                                                for (int l = 1; l < r; l++) {
                                                        c = t.charAt(l);
                                                        s.append(c);
                                                }
                                                for (int l = r + 1; l < t.length(); l++) {
                                                        c = t.charAt(l);
                                                        s2.append(c);
                                                }
                                        }
                                        n.preCond = s.toString();
                                        n.name = s2.toString();
                                }
                        }
                }
        }
}
```

```
/*
 * This method assigns the intraprocedural edges in the rcfg for a
 * message.We allready have the nodes in the rcfg vestor of a message, so
 * this function just iterates through the vector and assigns
 * intraprocedural edges to nodes contained within that Vector.The
 * interprocedural edges are allready obtained since every node in the rcfg
 * vector has also a rcfg vector, which correspond to the interprocedural
 * edges in the IRCFG
 *
 * @param n3 Nodes a tamporary object of type Nodes
 *
 * @param n2 Nodes a tamporary object of type Nodes
 */
@SuppressWarnings("unchecked")
public void deriveRCFGEdges() {
	for (int i = 0; i < v.size(); i++) {
		n = (Nodes) v.elementAt(i);
		if (n.type.equals("Message")) {
			for (int j = 0; j < n.rcfg.size(); j++) {
				Nodes n2 = (Nodes) n.rcfg.elementAt(j);
				for (int k = j + 1; k < n.rcfg.size(); k++) {
					Nodes n3 = (Nodes) n.rcfg.elementAt(k);
					if (n3.preCond.equals("")) {
			if (!(containsByName(n2.rcfgIntraproseduralEdge, n3))) {
				n2.rcfgIntraproseduralEdge.add(n3);
			}
			break;
			} else {
			if (!(containsByName(n2.rcfgIntraproseduralEdge, n3))) {
			n2.rcfgIntraproseduralEdge.add(n3);
						}
					}
				}
			}
		}
	}
}
```

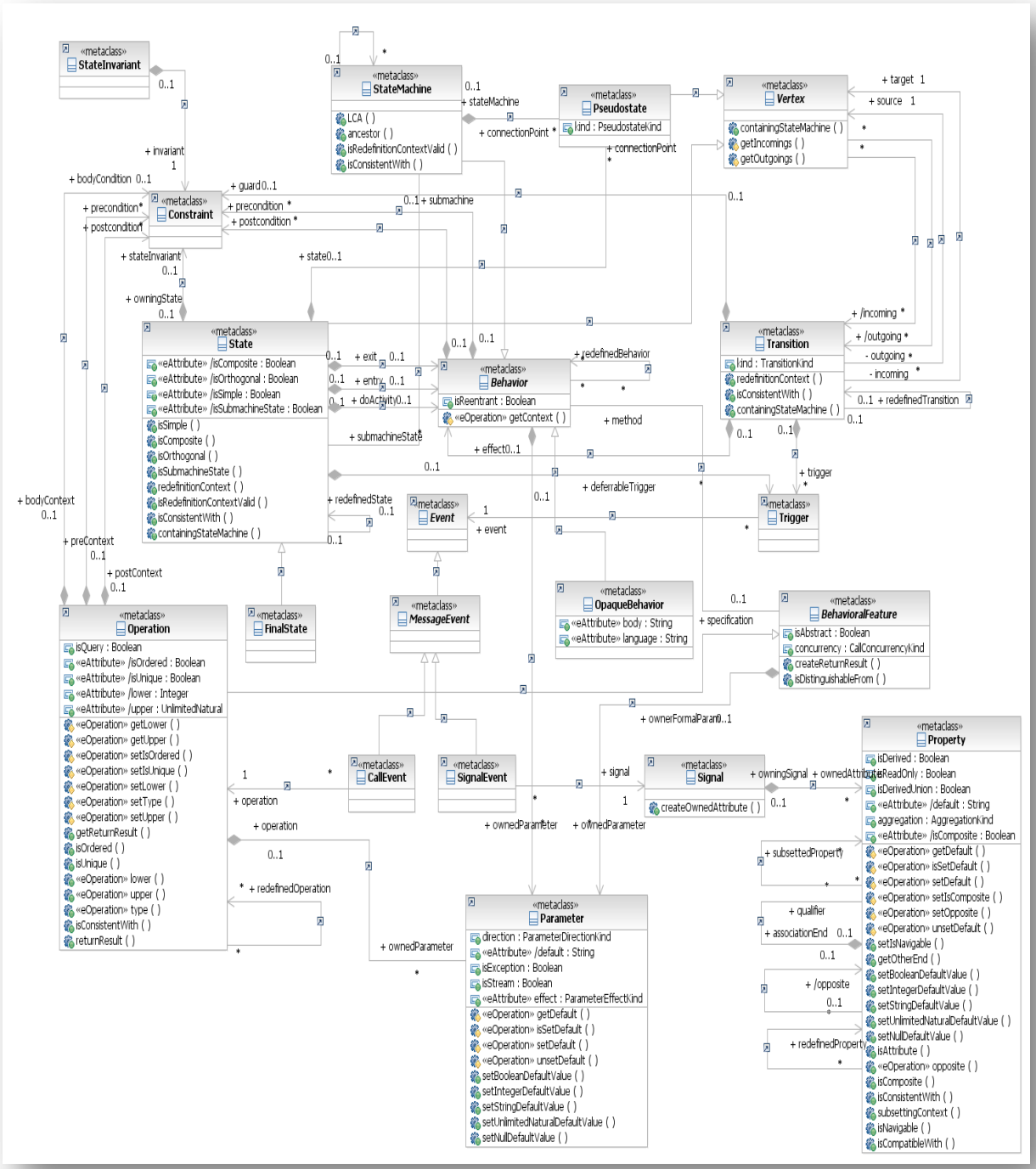
```
/*
 * This method may be spourious for the finished tool, but I found it useful
 * for every method to have an origin and destination. The method simply
 * assigns a name "External" to the method of which it can not find a
 * corresponding send class in the diagram. For example if a metodcall
 * invoken from outside of the diagram frame,its sendClassvariable is
 * assigne the value "External"
 *
 * @param n2 Nodes a tamporary object of type Nodes
 *
 * @param current String holds the name of current send-class
 *
 * @param external Boolean true if the sending class is external
 */
public void checkForExternalCall() {
	boolean external;
	String current;
	for (int i = 0; i < v.size(); i++) {
		n = (Nodes) v.elementAt(i);
		if (n.type.equals("Message")) {
			current = n.sendClass;
			external = true;
			for (int j = 0; j < v.size(); j++) {
				Nodes n2 = (Nodes) v.elementAt(j);
				if (n2.type.equals("Class")) {
					if (current.equals(n2.name)) {
						external = false;
					}
				}
			}
			if (external) {
				n.sendClass = "External";
			}
		}
	}
}
```

```
/*
 * This function iterates over the Vector holding Fragment objects and
 * derives values for their variables
 * @param f Fragment an object of type Fragment
 * @param f2 Fragment an object of type Fragment
 * @param n Nodes an object of type Nodes
 * @param s StringBuffer an object of type StringBuffer
 * @param s2 StringBuffer an object of type StringBuffer
 * @param index int holds the index of a character
 */
public void derivePropertiesForFragments() {
	StringBuffer s, s2;
	for (int i = 0; i < w.size(); i++) {
		Fragment f = (Fragment) w.elementAt(i);
		for (int j = 0; j < v.size(); j++) {
			Nodes n = (Nodes) v.elementAt(j);
			if (f.covered.equals(n.id)) {
				f.covered = n.name;
			}
			if (f.message.equals(n.id)) {
				f.message = n.name;
			}
		}
	}

	for (int i = 0; i < w.size(); i++) {
		Fragment f = (Fragment) w.elementAt(i);
		for (int j = 0; j < w.size(); j++) {
			Fragment f2 = (Fragment) w.elementAt(j);
			if (f.event.equals(f2.id)) {
				f.event = f2.name;
			}
			if (f.execution.equals(f2.id)) {
				f.execution = f2.name;
			}
			if (f.start.equals(f2.id)) {
				f.start = f2.name;
			}
			if (f.finish.equals(f2.id)) {
				f.finish = f2.name;
			}
		}
	}
	for (int i = 0; i < v.size(); i++) {
		Nodes n = (Nodes) v.elementAt(i);
		if (n.type.equals("Message")) {
			for (int j = 0; j < w.size(); j++) {
				Fragment f = (Fragment) w.elementAt(j);
				if (n.name.equals(f.message) && f.name.contains("Receive")) {
					s = new StringBuffer(f.name);
					s2 = new StringBuffer();
					int index = s.indexOf("_");
					for (int k = index + 1; k < s.length(); k++) {
						s2.append(s.charAt(k));
					}
					n.seq = Integer.parseInt(s2.toString());
				}
			}
		}
	}
```

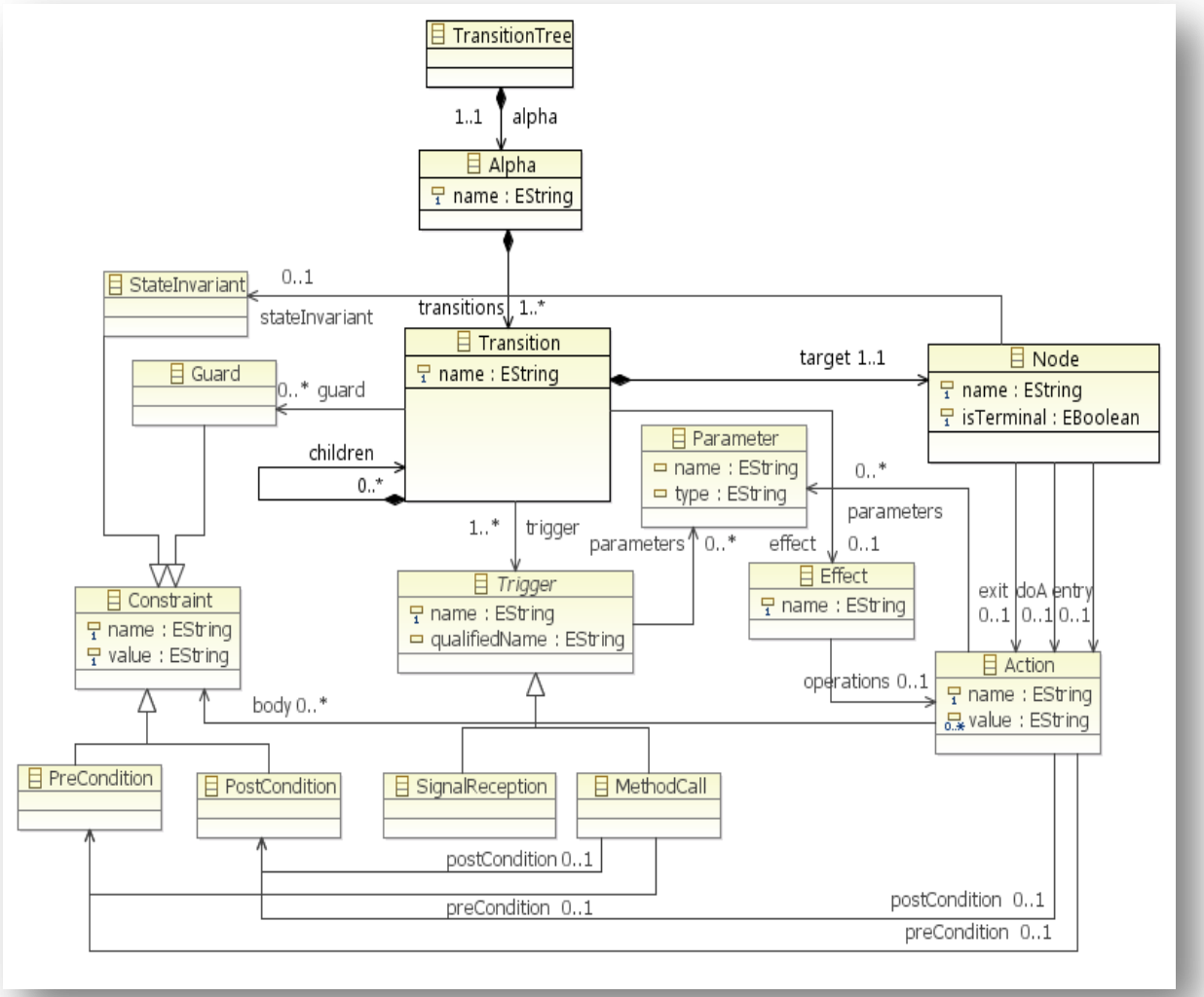
```
		}
}


/*
 * @param coveredBy Vector holds the Fragments corresponding to elements in
 * lifeline
 * @param lifelines Vector holds the elements contained in a lifeline (UML)
 * @param getsInvokation Vector holds the classes from where invokations
 * originate
 * @param invokes Vector holds the class in which we are making invokations
 * @param rcfg Vector holds the messages (Nodes) that can be invoken from the
 * messagebody
 * @param rcfgIntraproseduralEdge Vector holds the intraprocedural edges of a
 * node (message)
 * @param id String the id of the node
 * @param type String the type of the node (Message or Class)
 * @param preCond String a possible precondition for the method execution
 * @param sendClass String the name of class from where we can invoke the method
 * @param receiveClass String the name of originating class
 * @param seq int the sequence number of the node
 */
class Nodes {
		@SuppressWarnings("unchecked")
		Vector coveredBy = new Vector();
		@SuppressWarnings("unchecked")
		Vector lifelines = new Vector();
		@SuppressWarnings("unchecked")
		Vector getsInvokation = new Vector();
		@SuppressWarnings("unchecked")
		Vector invokes = new Vector();
		@SuppressWarnings("unchecked")
		Vector rcfg = new Vector();
		@SuppressWarnings("unchecked")
		Vector rcfgIntraproseduralEdge = new Vector();
		String id = "";
		String type = "";
		String name = "";
		String preCond = "";
		String sendClass = "";
		String receiveClass = "";
		int seq = -1;
}
```

```
/*
 * Class Fragments is a temporary solution, objects of this class are derived
 * from Fragments in UML terminology, as opposed to Classes and Operations that
 * are mapped to Nodes. Fragments can be Behavior Executions,Lifelines,Execution
 * Specification,Receive/ Send Events and so on.
 * @param id String the id of the Fragment
 * @param name String the name of the Fragment
 * @param type String the type of Fragment
 * @param covered String the owning Class
 * @param event String type of Event associated with Fragment
 * @param execution String the Execution Specification
 * @param message String the message propery of Fragment (if any!)
 * @param start String the start event (name of Fragment)
 * @param finish String the finish event (name of Fragment)
 * @param method String holds the methods contained in a Behavior Execution
 * @param mBody Vector holds the m events contained in the message body
 */
class Fragment {
	String id = "";
	String name = "";
	String type = "";
	String covered = "";
	String event = "";
	String execution = "";
	String message = "";
	String start = "";
	String finish = "";
	String method;
	@SuppressWarnings("unchecked")
	Vector mBody = new Vector();
}
```

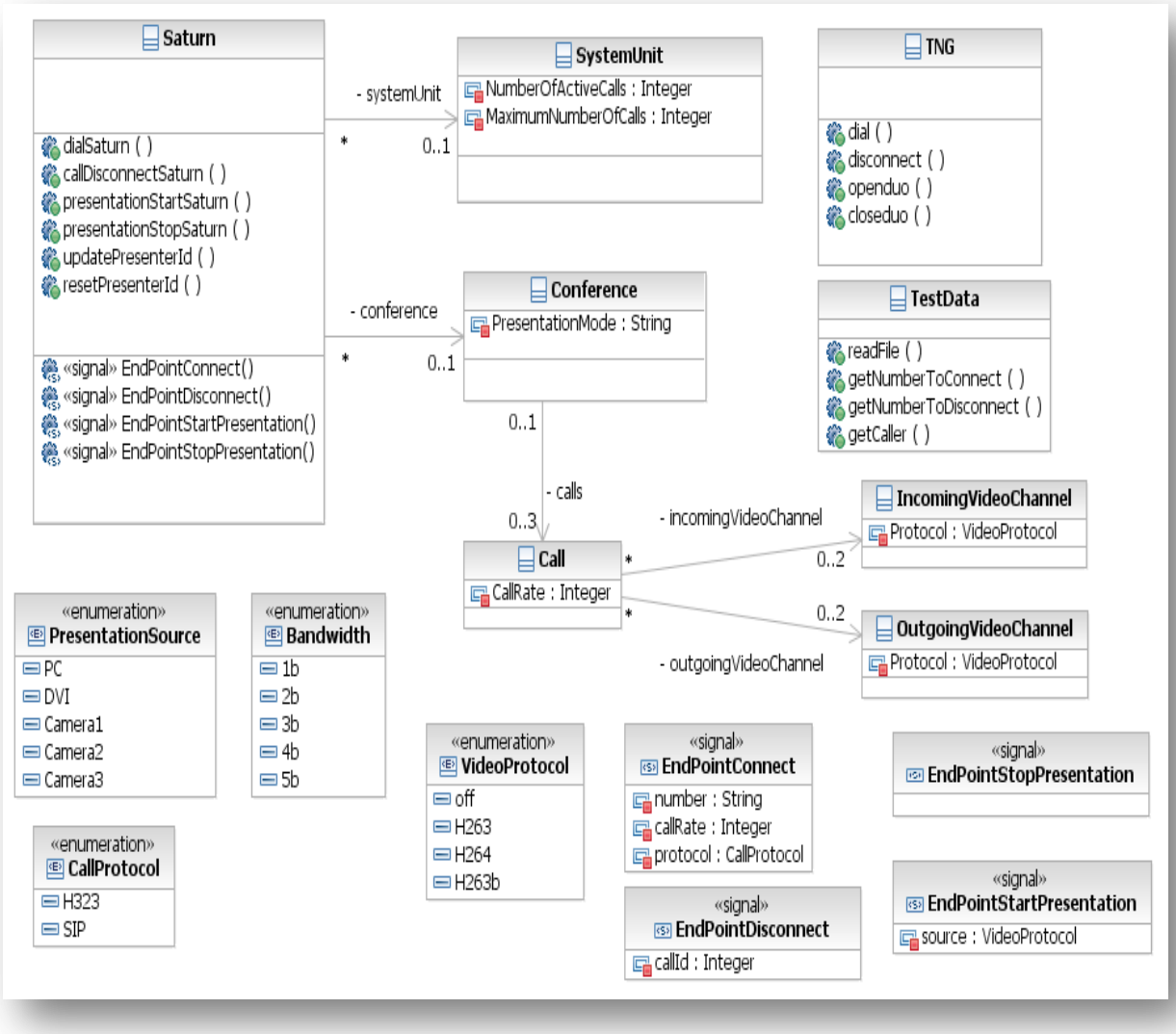
9.2 Appendix B – UML 2.0 Meta-Model (UML)



9.3 Appendix C – Transition Tree Meta-Model (EMF)



9.4 Appendix D – The domain model of Saturn (UML)



9.5 Appendix E – StateMachine2TransitionTree.atl (ATL)

```
--------------------------------------------------------------------
-- AUTHOR - Miran Damjanovic ---------------------------------------
-- DATE   - 30.03.2009 ---------------------------------------------
-- PLACE  - University of Oslo, Simula Research Laboratory ---------
-- EMAIL  - mirand@ifi.uio.no --------------------------------------
-- This m-2-m transformation transforms a UML State Machine diagram into
-- a Transition Tree diagram.
--------------------------------------------------------------------

module StateMachine2TransitionTree;
create OUT : transitiontree from IN : SM;

--------------------------------------------------------------------
-- HELPERS ---------------------------------------------------------
--------------------------------------------------------------------

-- This helper is actually a global variable holding the initial state
-- of the State Machine.
-- CONTEXT thisModule
-- RETURN: void
helper def : initial : SM!Pseudostate =
            SM!Pseudostate.allInstances()->select(s1|s1.kind=#initial)->first();

--------------------------------------------------------------------
-- RULES -----------------------------------------------------------
--------------------------------------------------------------------

-- Rule StateMachine2TransitionTree
-- This rule generates a transitiontree!TransitionTree along with its
-- "alpha" state from SM!StateMachine.
rule StateMachine2TransitionTree {
       from
           a: SM!StateMachine
       to
           b: transitiontree!TransitionTree(
                       alpha <- a.region->first().subvertex
                                       ->select(s|s.oclIsTypeOf(SM!Pseudostate))
                                       ->select(s1|s1.kind=#initial)->first()
                   )
}
```

```
-- Rule Pseudostate2Alpha
-- This rule generates a transitiontree!Alpha from a SM!Pseudostate. The rule
-- also checks that the produced Alpha is in fact the the initial state of
-- the State Machine. This check is in order to differentiate between the
-- different types of Pseudostates, e.g. Choice Pseudostate.
rule PseudoState2Alpha{
	from
			a: SM!Pseudostate  (a.oclIsTypeOf(SM!Pseudostate)and
				a.name = thisModule.initial.name)
		using{
		pstate : SM!Vertex = a.outgoing->first().target;
temp : Set(SM!Transition) = if pstate.oclIsTypeOf(SM!Pseudostate) then


		pstate.outgoing
				else
					a.outgoing
				endif;
			}
	to
		b: transitiontree!Alpha(
			name <- a.name,
				transitions <- temp->collect(t| if t.name <> a.name and	a.name =
				thisModule.initial.name
				then thisModule.Transition2OutgoingTransition(t,Set{b.name},
				Set{t.guard},t.trigger
				else
					transitiontree!Transition
				endif

					)
				)
}
```

```
-- Rule Transition2OutgoingTransition
-- This rule is a Called Rule, and is recursive. The rule is first called
-- from rule PseudoState2Alpha. The recursion stops when we come to a
-- state we have visited earlier, these states are passed along in the
-- set "p". The guard and trigger are passed along in order to provide
-- smooth transformation and merging when dealing with Pseudostates, or
-- more precisely a Choice Pseudostate. In case of a Choice, the trigger
-- remains the same when merged, but the guards are added to the same set.
-- This transformation generates several transitiontree!Outgoingtransitions
-- along with their targets and children from SM!Transitions. Children
-- derived from SM!Transitions that can be taken from the target state
-- of the parent SM!Transition.
rule Transition2OutgoingTransition(a : SM!Transition,p : Set(String), guard      : Set(SM!Constraint),trigger :
SM!Trigger){
                    using{
                    temp : Set(SM!Transition) = a.target.outgoing;
      parents : Set(String) = p;
                }
      to
          b: transitiontree!Transition(
          name <- a.name,
                              guard <- guard,
                              trigger <- trigger->collect(t|t.event),
                              effect <- a.effect,
                              children <- temp->collect(t | if not parents-> includes(a.target.name)  then
            if t.target.oclIsTypeOf(SM!Pseudostate) then
            t.target.outgoing->collect(s|
                    if s.target.name = a.target.name then
          thisModule.Transition2OutgoingTransition(s,p->        including(t.name),Set{s.guard,t.guard},t.trigger)
                              else thisModule.Transition2OutgoingTransition(s,parents,Set{s.guard,t.guard},
t.trigger)
                              endif      )
                        else
                    thisModule.Transition2OutgoingTransition(t,parents->
including(a.target.name),Set{t.guard},t.trigger)
              endif
                    else
                              transitiontree!Transition
                    endif),
              target <- state),
        state : transitiontree!Node (
          name <- a.target.name,
                    doA <-  a.target.doActivity,
                    entry <- a.target.entry,
                    exit <- a.target.exit,
                    stateInvariant <-  a.target.ownedElement.first(),
                    isTerminal <-      if a.target.oclIsTypeOf(SM!FinalState) then
                    true
           else
                                                        false
                                                endif
                                                )
                    do{b;}
}
```

```
-- Rule Constraint2Constraint
-- This rule generates a transitiontree!Constraint from a
-- SM!Constraint, along with its name and value properties.

rule Constraint2Constraint{
	from
		a:SM!Constraint
	to
		b:transitiontree!Constraint(
			name <- a.name,
			value <- a.specification.stringValue()
		)
}

-- Rule Parameter2Parameter
-- This rule generates a transitiontree!Parameter from a
-- SM!Parameter, along with its type and name properties.

rule Parameter2Parameter{
	from
		a: SM!Parameter
	to
		b:transitiontree!Parameter(
			name <- a.name,
			type <- a.type.name
		)

}

-- Rule Property2Parameter
-- This rule generates a transitiontree!Parameter from a
-- SM!Property, along with its type and name properties.
rule Property2Parameter{
	from
		a: SM!Property
	to
		b:transitiontree!Parameter(
			name <- a.name,
			type <- a.type.name
		)
}

-- Rule CallEvent2MethodCall
-- This rule generates a transitiontree!CallEvent from a
-- SM!CallEvent, along with its name,qualifiedName,preCondition,
-- postCondition and parameters properties.
rule CallEvent2MethodCall{
	from
		a:SM!CallEvent
	to
		b:transitiontree!MethodCall(
			name <- a.operation.name,
			qualifiedName <- a.operation.qualifiedName,
			preCondition <- a.operation.precondition->first(),
			postCondition <- a.operation.postcondition->first(),
			parameters <- a.operation.ownedParameter
				)
}
```

```
-- Rule SignalEvent2SignalReception
-- This rule generates a transitiontree!SignalReception from a
-- SM!SignalEvent, along with its name, qualifiedName and
-- parameters properties.

rule SignalEvent2SignalReception{
	from
		a:SM!SignalEvent
	to
		b:transitiontree!SignalReception(
			name <- a.signal.name,
			qualifiedName<- a.signal.qualifiedName,
			parameters <- a.signal.ownedAttribute)
}

-- Rule OpaqueBehavior2Effect
-- This rule generates a transitiontree!Effect from a
-- SM!OpaqueBehavior, along with its name and operations
-- properties. We differentiate here beetween OpaqueBehaviors
-- that generate Effects according to weather its specification
-- property is set.

rule OpaqueBehavior2Effect{
	from
		a:SM!OpaqueBehavior(not a.specification.oclIsUndefined())
	to
		b:transitiontree!Effect(
				name <- a.name,
				operations<- a.specification)
}


-- Rule OpaqueBehavior2Action

-- This rule generates a transitiontree!Action from a
-- SM!OpaqueBehavior, along with its name,value,preCondition,
-- postCondition and parameters properties.We differentiate
-- here beetween OpaqueBehaviors that generate Action according
-- to weather its specification property is not set. The
-- transitiontree!Action generated from this rule is associated
-- with do/entry/exit-actions of a transitiontree!Node.

rule OpaqueBehavior2Action{
	from
		a:SM!OpaqueBehavior(a.specification.oclIsUndefined())
	to
		b:transitiontree!Action(
			name <- a.name,
			value <- a.body,
			preCondition <- a.precondition->first(),
			postCondition <- a.postcondition->first(),
			parameters <- a.ownedParameter
			)
}
```

```
-- Rule Operation2Action
-- This rule generates a transitiontree!Action from a
-- SM!Operation, along with its name,preCondition,postCondition
-- and parameters properties. The transitiontree!Action generated
-- from this rule is associated with operations property of a
-- transitiontree!Effect.

rule Operation2Action{
	from
		a:SM!Operation (not a.oclIsUndefined() and a.oclIsTypeOf(SM!Operation))
	to
		b:transitiontree!Action(
			name <- a.name,
			preCondition <- a.precondition->first(),
			postCondition <- a.postcondition->first(),
			parameters <- a.ownedParameter
			)

}
```

9.6 Appendix F - The generated Transition Tree (XML)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:transitiontree="http://transitiontree/1.0">
  <transitiontree:TransitionTree>
    <alpha name="Start">
      <transitions name="Start->Idle">
        <children name="Idle->Connected" guard="/11" trigger="/61 /68">
          <children name="Connected->NotFull" guard="/7" trigger="/64 /71">
            <children name="NotFull->Connected" guard="/2" trigger="/72 /58">
              <target name="Connected" stateInvariant="/13"/>
            </children>
            <children name="NotFull->Full" guard="/5" trigger="/60 /74">
              <children name="Full->NotFull" guard="/8" trigger="/75 /57">
                <target name="NotFull" stateInvariant="/6"/>
              </children>
              <target name="Full" stateInvariant="/15"/>
            </children>
            <children name="NotFull->NotFull_1" guard="/10" trigger="/59 /76">
              <target name="NotFull" stateInvariant="/6"/>
            </children>
            <children name="NotFull->NotFull_2" guard="/4" trigger="/56 /77">
              <target name="NotFull" stateInvariant="/6"/>
            </children>
            <target name="NotFull" stateInvariant="/6"/>
          </children>
          <children name="Connected->Terminated" trigger="/55 /73">
            <target name="Terminated" isTerminal="true"/>
          </children>
          <target name="Connected" stateInvariant="/13"/>
        </children>
        <target name="Idle" stateInvariant="/14"/>
      </transitions>
    </alpha>
  </transitiontree:TransitionTree>
  <transitiontree:Constraint name="full" value="self.systemUnit.numberOfActiveCalls
=  self.systemUnit.maximumNumberOfCalls  and self.conference.presentationMode =
'off' and self.conference.calls->select(c:Call |
c.incomingPresentationChannel.protocol != VideoProtocol::off or
c.outgoingPresentationChannel.protocol != VideoProtocol::off)->size() =0  "/>
  <transitiontree:Constraint value="self.systemUnit.NumberOfActiveCalls = 2"/>
  <transitiontree:Constraint name="connected"
value="self.systemUnit.numberOfActiveCalls =1 and self.conference.presentationMode
= 'off' and self.conference.calls->select(c:Call |
c.incomingPresentationChannel.protocol != VideoProtocol::off or
c.outgoingPresentationChannel.protocol != VideoProtocol::off)->size() =0  "/>
  <transitiontree:Constraint value="self.systemUnit.NumberOfActiveCalls > 1 and
self.systemUnit.NumberOfActiveCalls &lt; self.systemUnit.MaximumNumberOfCalls -1"/>
  <transitiontree:Constraint value="self.systemUnit.NumberOfActiveCalls =
self.systemUnit.MaximumNumberOfCalls -1"/>
  <transitiontree:Constraint name="Not_Full"
value="(self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls&lt;self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol &lt;> VideoProtocol::Off or
c.outgoingVideoChannel->asSequence()->last().Protocol &lt;> VideoProtocol::Off)-
>size()=0"/>
  <transitiontree:Constraint value="self.systemUnit.NumberOfActiveCalls = 1"/>
  <transitiontree:Constraint name="Guard_Full->NotFull"
value="self.systemUnit.NumberOfActiveCalls = self.systemUnit.MaximumNumberOfCalls -
2"/>
  <transitiontree:Constraint name="idle" value="self.systemUnit.numberOfActiveCalls
= 0 and self.conference.presentationMode = 'off'"/>
```

```
  <transitiontree:Constraint value="self.systemUnit.NumberOfActiveCalls >2 and
self.systemUnit.NumberOfActiveCalls &lt; self.systemUnit.MaximumNumberOfCalls -1"/>
  <transitiontree:Constraint name="" value="self.systemUnit.NumberOfActiveCalls =
0"/>
  <transitiontree:Constraint name="not_full"
value="(self.systemUnit.numberOfActiveCalls  > 1 and
self.systemUnit.numberOfActiveCalls  &lt; self.systemUnit.maximumNumberOfCalls )and
self.conference.presentationMode = 'off' and self.conference.calls->select(c:Call |
c.incomingPresentationChannel.protocol != VideoProtocol::off or
c.outgoingPresentationChannel.protocol != VideoProtocol::off)->size() =0
&#xD;&#xA;"/>
  <transitiontree:Constraint name="Connected"
value="self.systemUnit.NumberOfActiveCalls=1 and self.conference.PresentationMode =
'Off' and self.conference.calls->select(c:Call | c.incomingVideoChannel-
>asSequence()->last().Protocol &lt;> VideoProtocol::Off or c.outgoingVideoChannel-
>asSequence()->last().Protocol &lt;> VideoProtocol::Off)->size() =0  "/>
  <transitiontree:Constraint name="IdleStateInvariant"
value="self.systemUnit.NumberOfActiveCalls = 0 and self.conference.PresentationMode
= 'Off'"/>
  <transitiontree:Constraint name="Full"
value="self.systemUnit.NumberOfActiveCalls=self.systemUnit.MaximumNumberOfCalls
and self.conference.PresentationMode = 'Off' and self.conference.calls-
>select(c:Call | c.incomingVideoChannel->asSequence()->last().Protocol &lt;>
VideoProtocol::Off or c.outgoingVideoChannel->asSequence()->last().Protocol &lt;>
VideoProtocol::Off)->size() =0  "/>
  <transitiontree:Parameter name="number" type="String"/>
  <transitiontree:Parameter name="protocol" type="VideoProtocol"/>
  <transitiontree:Parameter name="bandwidth" type="VideoProtocol"/>
  <transitiontree:Parameter name="nameOfCaller" type="String"/>
  <transitiontree:Parameter name="number" type="String"/>
  <transitiontree:Parameter name="name" type="String"/>
  <transitiontree:Parameter name="nameOfCaller" type="String"/>
  <transitiontree:Parameter name="number" type="Integer"/>
  <transitiontree:Parameter name="nameOfCaller" type="String"/>
  <transitiontree:Parameter name="command" type="String"/>
  <transitiontree:Parameter name="name" type="String"/>
  <transitiontree:Parameter name="number" type="String"/>
  <transitiontree:Parameter name="source" type="VideoProtocol"/>
  <transitiontree:Parameter name="protocol" type="CallProtocol"/>
  <transitiontree:Parameter name="bandwidth" type="Bandwidth"/>
  <transitiontree:Parameter name="nameOfFile" type="String"/>
  <transitiontree:Parameter name="systemUnit" type="SystemUnit"/>
  <transitiontree:Parameter name="outgoingVideoChannel"
type="OutgoingVideoChannel"/>
  <transitiontree:Parameter name="NumberOfActiveCalls" type="Integer"/>
  <transitiontree:Parameter name="attribute2" type="String"/>
  <transitiontree:Parameter name="calls" type="Call"/>
  <transitiontree:Parameter name="attribute1" type="Integer"/>
  <transitiontree:Parameter name="incomingVideoChannel"
type="IncomingVideoChannel"/>
  <transitiontree:Parameter name="source" type="VideoProtocol"/>
  <transitiontree:Parameter name="callRate" type="Integer"/>
  <transitiontree:Parameter name="" type="Call"/>
  <transitiontree:Parameter name="callId" type="Integer"/>
  <transitiontree:Parameter name="" type="Call"/>
  <transitiontree:Parameter name="PresentationMode" type="String"/>
  <transitiontree:Parameter name="CallRate" type="Integer"/>
  <transitiontree:Parameter name="" type="Saturn"/>
  <transitiontree:Parameter name="OWNER_SATURN" type="Saturn"/>
  <transitiontree:Parameter name="Protocol" type="VideoProtocol"/>
  <transitiontree:Parameter name="MaximumNumberOfCalls" type="Integer"/>
  <transitiontree:Parameter name="number" type="String"/>
  <transitiontree:Parameter name="conference" type="Conference"/>
  <transitiontree:Parameter name="protocol" type="CallProtocol"/>
  <transitiontree:Parameter name="Protocol" type="VideoProtocol"/>
  <transitiontree:Parameter name="" type="Conference"/>
```

```
  <transitiontree:MethodCall name="callDisconnectSaturn" qualifiedName="Blank
Package::Saturn::callDisconnectSaturn" parameters="/23"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:MethodCall name="callDisconnectSaturn" qualifiedName="Blank
Package::Saturn::callDisconnectSaturn" parameters="/23"/>
  <transitiontree:MethodCall name="callDisconnectSaturn" qualifiedName="Blank
Package::Saturn::callDisconnectSaturn" parameters="/23"/>
  <transitiontree:MethodCall name="callDisconnectSaturn" qualifiedName="Blank
Package::Saturn::callDisconnectSaturn" parameters="/23"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:MethodCall name="presentationStopSaturn" qualifiedName="Blank
Package::Saturn::presentationStopSaturn"/>
  <transitiontree:MethodCall name="presentationStopSaturn" qualifiedName="Blank
Package::Saturn::presentationStopSaturn"/>
  <transitiontree:MethodCall name="dialSaturn" qualifiedName="Blank
Package::Saturn::dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:SignalReception name="EndPointConnect" qualifiedName="Blank
Package::EndPointConnect" parameters="/50 /40 /52"/>
  <transitiontree:SignalReception name="EndPointConnect" qualifiedName="Blank
Package::EndPointConnect" parameters="/50 /40 /52"/>
  <transitiontree:SignalReception name="EndPointConnect" qualifiedName="Blank
Package::EndPointConnect" parameters="/50 /40 /52"/>
  <transitiontree:SignalReception name="EndPointConnect" qualifiedName="Blank
Package::EndPointConnect" parameters="/50 /40 /52"/>
  <transitiontree:SignalReception name="EndPointDisconnect" qualifiedName="Blank
Package::EndPointDisconnect" parameters="/42"/>
  <transitiontree:SignalReception name="EndPointDisconnect" qualifiedName="Blank
Package::EndPointDisconnect" parameters="/42"/>
  <transitiontree:SignalReception name="EndPointConnect" qualifiedName="Blank
Package::EndPointConnect" parameters="/50 /40 /52"/>
  <transitiontree:SignalReception name="EndPointDisconnect" qualifiedName="Blank
Package::EndPointDisconnect" parameters="/42"/>
  <transitiontree:SignalReception name="EndPointDisconnect" qualifiedName="Blank
Package::EndPointDisconnect" parameters="/42"/>
  <transitiontree:SignalReception name="EndPointConnect" qualifiedName="Blank
Package::EndPointConnect" parameters="/50 /40 /52"/>
  <transitiontree:Action name="getNumberToConnect" parameters="/21 /22"/>
  <transitiontree:Action name="callDisconnectSaturn" parameters="/23"/>
  <transitiontree:Action name="dialSaturn" parameters="/16 /17 /18"/>
  <transitiontree:Action name="presentationStartSaturn" parameters="/28"/>
  <transitiontree:Action name="disconnect" parameters="/20"/>
  <transitiontree:Action name="dial" parameters="/27 /29 /30"/>
  <transitiontree:Action name="getNumberToDisconnect" parameters="/26 /19"/>
  <transitiontree:Action name="readFile" parameters="/31"/>
  <transitiontree:Action name="presentationStopSaturn"/>
  <transitiontree:Action name="updatePresenterId"/>
  <transitiontree:Action name="resetPresenterId"/>
  <transitiontree:Action name="getCaller" parameters="/24 /25"/>
  <transitiontree:Action name="openduo"/>
  <transitiontree:Action name="closeduo"/>
</xmi:XMI>
```

9.7 Appendix G – SM2TT.m2t (MOFScript)

```
/**
 * @author Miran Damjanovic
 * @date 12.03.2009
 * This m-2-t transformation generates test-scripts with inserted
 * test-data and TNG-grammar from the Transition Tree diagram.Each
 * distinct path in the Transition Tree is transformed into sequences
 * of method-calls specified in the script.
 */

 /**
 * Text-transformation SM2TT. Has a main rule that starts the transformation
 * process by mapping the Alpha state (initial state) first, and from there
 * mapping each transition and their children (transitions).
 * @param uml in, the input model (class diagram)
 * @param transitiontree in, the input model (transition tree)
 * @param enumerationList List, list of enumerations derived from the input model
 * @param counter Integer, clobal counter used in the generated file names
 * @param jClass String, the pointer for the java class invoked in the TNG-scripts
 * @param guardValue String, the variable holding the current guard-value
 */
texttransformation SM2TT (in transitiontree: "http://transitiontree/1.0",
                                              in uml:"http://www.eclipse.org/uml2/2.0.0/UML" )
{
        var counter : Integer = 0;
        var enumerationList : List;
        var jClass : String = "t.";
        var javaPath : String = "C:/Documents and Settings/Damjanovic
        Miran/IBM/rationalsdp/workspace/IntGenerator/";
        var guardValue : String;

/**
 * rule main starts the transformation process by invoking the start() rule for the
 * initial state (Alpha) of the transition tree. It also computes the enumeration
 * list.
 * @param fileList List, list of elements to be printed to file
 */
        transitiontree.TransitionTree::main(){
                        var fileList : List; //list of output elements

                        uml.objectsOfType(uml.Enumeration)->forEach(e : uml.Enumeration){
                                enumerationList.add(e);
                        }
                        if(transitiontree.Alpha != null){
                        transitiontree.alpha.start(fileList); // start first mapping
                }
        }

/**
 * rule start maps the initial state and starts mapping its child, the first
 * transition in the transition tree.
 */
transitiontree.Alpha::start(fileList : List){
                fileList.add(getImportStatements());
                fileList.add("# Initial State: " + self.name);
                self.transitions->forEach(trans : transitiontree.Transition){
                fileList.addAll("# Transition: " + trans.name + "\tGuard: " +
                trans.guard.first().value);
                trans.mapTransition(fileList); //for each outgoing, map transition
                }
        }
```

```
/**
 * rule mapTransition generates output associated with each transition and its
 * associated elements like triggers, guards and targets. This rule is recursive,
 * and is called for each oh the transitions children, if any. The recursive behavior
 * ends when the current transition does not have any more children (transitions). At
 * the point the recursion stops, the output is printed to file. One file will be
 * generated for each path in the transition tree.
 * @param fileList List, list of elements to be printed to file
 * @param f file, the generated output file
 * @param generatedInteger Integer, the randomly generated integer (from Java)
 * @param i Integer, temporary helper variable
 */
transitiontree.Transition::mapTransition(fileList : List){       //map Transition Metaclass
              var generatedInteger : Integer;
              var i :Integer = 0;

              if(not self.trigger.isEmpty()){
              generatedInteger = java
              ("RandomIntGenerator","getRandomInt",self.trigger.size(),javaPath);
              fileList.addAll("# Transition: " + self.name + "\tGuard: " +
              self.guard.first().value);
              self.trigger->forEach(t : transitiontree.Trigger){
               if(i = generatedInteger){
                   guardValue =  parseEnumeration(self.guard.first().value);
                   fileList.add("# Start Trigger: " + t.name + ", Type: " + t.oclGetType());
                   fileList.add("eval = '" + t.name+"'\n");
                   fileList.add(t.mapTrigger(self.guard.first().value));
                   stdout.println("Trigger for: " + self.name + ", TriggerName: " + t.name);
                   fileList.add("# Finish Trigger: " + t.name + ", Type: " + t.oclGetType());
                 }
                 i += 1;
               }
              }

              if(self.effect != null){ //check for owned Effect
                fileList.add("# Start Effect: " + self.effect.name);
                fileList.add(self.effect.mapEffect());
                fileList.add("# Finish Effect: " + self.effect.name);
              }

              fileList.add(self.target.mapNode());

              if(self.children.isEmpty()){
                     file f("TestCase_" + counter + ".ttr");
                     stdout.println("\tWriting file " + "TestCase_" + counter + ".ttr");
                     fileList.add("print(\"Evaluation of OCL-expressions:\")");
                     fileList.add("for res in results:");
                     fileList.add("\tprint(res)");

                     fileList->forEach(c){
                            println(c);
                     }
                     println("%>");
                     println("\ttesttarget.disconnect\n");
                     counter+=1;
              }
              else{
                self.children->forEach(child : transitiontree.Transition){
                  var fileListChild : List;
                  fileList->forEach(c){
                      fileListChild.add(c);
                  }
                      child.mapTransition(fileListChild)

                      }
              }
       }
```

```
/**
 * rule mapTrigger generates output associated with each trigger and its
 * associated properties. Each trigger is either a MethodCall or a
 * SignalReception.
 * @param oldGuardValue String, the value of the guard prior to changes
 * @param listTrigger List, the list of output statements for the trigger
 * @param enum String, variable holding the enumeration type of a parameter
 * @param param String, temporary helper variable
 * @param relationships String, holding the relationship of a trigger (if any)
 * @param m transitiontree.MethodCall, variable of type transitiontree.MethodCall
 * @param generatedInteger Integer, the randomly generated integer (from Java)
 * @param i Integer, temporary helper variable
 * @return ret String, the total generated output for a trigger as String
 */
  transitiontree.Trigger::mapTrigger(oldGuardValue : String ){ //map Trigger Metaclass
              var listTrigger : List;
              var ret : String;
              var enum : String;
              var param : String;
              var relationships : String;
              var m : transitiontree.MethodCall;
              var i :Integer = 0;
              var generatedInteger : Integer;

              if(self.oclGetType().equals("MethodCall")){ // if Trigger is of type MethodCall
                     m = self;
                     relationships = hasRelationships(m.qualifiedName);
                     if(relationships.equals("NoRelationships")){
                     param = "#" + "testtarget." + m.name + " ";
                     if(not m.parameters.isEmpty()){
                               m.parameters->forEach(p:transitiontree.Parameter){
                                      if(isEnumeration(p.type)){
                                          enum = getEnumeration(p.type);
                                              param +=  enum + " ";
                                              checkGuard(p.name,enum);
                                      }
                                      if(p.type.equals("Integer")){
                                     generatedInteger = java
                                     ("RandomIntGenerator","getRandomInt",10,javaPath) + " ";
                                          param += generatedInteger;
                                          checkGuard(p.name,generatedInteger);
                                      }
                                   if(p.equals("String")){
                                              param += p.name + " ";
                                  }
                                    }
                     }
                         listTrigger.add(param);
                     }
                     else{
                        listTrigger.add(mapRelationships(relationships,"MethodCall"));
                     }
              }
              if(self.oclGetType().equals("SignalReception")){ // if Trigger is of type
SignalReception
                     param =  self.name + " ";
                     relationships = hasRelationships(self.qualifiedName);
                     if(relationships.equals("NoRelationships")){  // check if
SignalReception has relationships
                         if(not self.parameters.isEmpty()){   //has owned Parameters
                               self.parameters->forEach(p:transitiontree.Parameter){
                                      if(isEnumeration(p.type)){
                                         enum = getEnumeration(p.type);
                        param +=  enum + " ";
                        checkGuard(p.name,enum);
                     }
```

```
                    if(p.type.equals("Integer")){
                    generatedInteger = java
                    ("RandomIntGenerator","getRandomInt",10,javaPath) + " ";
                        param += generatedInteger;
                        checkGuard(p.name,generatedInteger);
                                    }
                    if(p.type.equals("String")){
                      param += p.name + " ";
                              }
                          }
                        }
                        listTrigger.add(param);
                    }
                      else{
                       listTrigger.add(mapRelationships(relationships,"SignalReception"));
                    }
                }
                if(not oldGuardValue.equals("")){
                 listTrigger.addAllFirst(updateAttList(oldGuardValue.trim()));
                }
                if(not listTrigger.isEmpty()){
                        listTrigger->forEach(t : String){
                                ret += t + "\n";
                        }
                }
                return  ret.trim();
        }
/**
 * rule checkGuard checks if an expression contains an attribute name,
 * if so the expression (global guardValue) is altered such that the generated
 * value for that attribute is substituted in the expression insted of the attribute name.
 * @param i Integer, temporary helper variable
 * @param pname String, the name of the attribute
 * @param sub String, the substitution value
 * @param value String, temporary helper variable, holds the guard value
 * @param lowerChar String, temporary helper variable
 * @param upperChar String, temporary helper variable
 * @param lower Integer, temporary helper variable
 * @param upper Integer, temporary helper variable
 */
        module::checkGuard(pName : String, sub : String){
          var value : String =  guardValue;
          var lowerChar : String;
          var upperChar : String;
          var i : Integer;
      var lower : Integer;
      var upper : Integer;

      i = value.indexOf(pName);
      while(i != -1){
        upper = i+pName.size();
        upperChar = value.charAt(upper);
        if(i==0){
          lowerChar = " ";
        }
        else{
            lower = i-1;
            lowerChar = value.charAt(lower);
        }
        if(checkCharList(lowerChar) and checkCharList(upperChar)){
            value = value.substringBefore(pName) + sub + value.substringAfter(pName);
            i = value.indexOf(pName);
        }
        else{
            i = -1;
        }
      }
      guardValue = value.trim();
 }
```

```
/**
 * rule checkCharList is a helper rule that checks if a character is
 * one of the types given in a character list.
 * @param charList List, list of characters
 * @return found Boolean, boolean return value
 */
       module::checkCharList(char : String){
         var charList : List;
         var found : Boolean = false;
      charList.add(" ");
      charList.add("<");
      charList.add(">");
      charList.add("=");
      charList.add("(");
      charList.add(")");
      charList->forEach(c : String){
        if(c.equals(char)){
            found = true;
            break;
        }
      }
      return found;
       }
/**
 * rule mapEffect generates statements for an effect. The effect expression
 * is evaluated in the OCL-evaluator (IEOS)
 * @param listEffect String, contains statements for the effect
 * @param listOp List, list of operations associated with the effect
 */
       transitiontree.Effect::mapEffect(){
              var listEffect : String;
              var listOp : List;

              if(self.operations==null){ //Effect has no owned Operations
                     listEffect="# Effect " + self.name + " has no specification.";
              }
              else{
                  if(self.operations.preCondition!=null){
                     listEffect += "# PreCondition of Efect: " +
                     self.operations.preCondition.name + "\n";
                     listEffect += "queryResult = c.updateObjectDiagram([],\"" +
                     self.operations.preCondition.value + "\")\n";
                     listEffect += "print \"Result of query: \", queryResult\n";
                  }

                     listOp = findOperation(self.operations.name);
                     listOp->forEach(el : String){
                        listEffect += el;
                     }

                     if(self.operations.postCondition!=null){
                     listEffect += "\n# PostCondition of Effecr: " +
                     self.operations.postCondition.name;
                     listEffect += "\nqueryResult = c.updateObjectDiagram([],\"" +
                     self.operations.postCondition.value + "\")\n";
                     listEffect += "print \"Result of query: \", queryResult\n";
              }
                 }
              return listEffect;
       }
```

```
/**
 * rule mapNode produces the output for an instance of the Node
 * metaclass specified in the Transition Tree metamodel.More specifically
 * the rule calls the mapStateInvariant() rule in case the node has
 * a StateInvariant, and also checks if the node has do/entry/exit Actions.
 * @param listNode List, the list of output elements for the node
 * @param val String, holds the values for do/exit/entry-activities
 * @return ret String, the return statements with output for the node
*/
        transitiontree.Node::mapNode(){
          var listNode : List;
          var ret: String;
          var val: String;

          listNode.add("# State: " + self.name);
          if(self.stateInvariant!=null){
            listNode.add(self.stateInvariant.mapStateInvariant());
          }

          if(self.entry != null){
            val = self.entry.value.first();
            listNode.add("# Start Entry-Activity : " + val);
            listNode.add("# PreCondition for Entry-Activity: " + findOperationsPreCond(val));
         listNode.add("\nqueryResult = c.updateObjectDiagram([],\"" +
        findOperationsPreCond(val) + "\")");
         listNode.add( "print \"Result of query: \", queryResult");
            listNode.addAll(findOperation(val));
            listNode.add("# PostCondition for Entry-Activity: " + findOperationsPostCond(val));
         listNode.add("\nqueryResult = c.updateObjectDiagram([],\"" +
        findOperationsPostCond(val) + "\")");
         listNode.add("print \"Result of query: \", queryResult");
            listNode.add("#Finished Entry-Activity : " + val);
          }

         if(self.doA != null){
         val = self.doA.value.first();
         listNode.add("# Start Do-Activity : " + val);
         listNode.add("# PreCondition for Do-Activity: " + findOperationsPreCond(val));
         listNode.add("\nqueryResult = c.updateObjectDiagram([],\"" +
        findOperationsPreCond(val) + "\")");
         listNode.add("print \"Result of query: \", queryResult");
         listNode.addAll(findOperation(val));
         listNode.add("# PostCondition for Do-Activity: " +findOperationsPostCond(val));
         listNode.add("\nqueryResult = c.updateObjectDiagram([],\"" + f
        indOperationsPostCond(val) + "\")");
         listNode.add("print \"Result of query: \", queryResult");
                listNode.add("# Finished Do-Activity : " + val);
          }

          if(self.exit != null){
         val = self.exit.value.first();
         listNode.add("# Start Exit-Activity : " + val);
         listNode.add("# PreCondition for Exit-Activity: " + findOperationsPreCond(val));
         listNode.add("\nqueryResult = c.updateObjectDiagram([],\"" +
        findOperationsPreCond(val) + "\")");
         listNode.add("print \"Result of query: \", queryResult");
         listNode.addAll(findOperation(val));
         listNode.add("# PostCondition for Exit-Activity: " + findOperationsPostCond(val));
         listNode.add("\nqueryResult = c.updateObjectDiagram([],\"" +
        findOperationsPostCond(val) + "\")");
         listNode.add("print \"Result of query: \", queryResult");
         listNode.add("# Finished Exit-Activity : " + val);
          }
          listNode->forEach(n : String){
                        ret += n + "\n";
                }

                return ret.trim();       //return output
        }
```

```
/**
 * rule findOperationsPrecondition computes the precondition
 * associated with an operation.
 * @param opName String, the name of the operation
 * @return ret String, the precondition of the operation as String
 */
       module::findOperationsPreCond(opName : String){
         var ret : String = " ";
         uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
          c.ownedOperation->forEach(op : uml.Operation){
              if(opName.equals(op.name)){
                ret = op.precondition.first().name;
              }
          }
         }
         return ret;
       }
/**
 * rule findOperationsPostcondition computes the postcondition
 * associated with an operation.
 * @param opName String, the name of the operation
 * @return ret String, the postcondition of the operation as String
 */
       module::findOperationsPostCond(opName : String){
         var ret : String = " ";
      uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
          c.ownedOperation->forEach(op : uml.Operation){
              if(opName.equals(op.name)){
                ret = op.postcondition.first().name;
              }
          }
      }
      return ret;
    }
/**
 * rule findOperation checks if an operation has another operation
 * specified in its specification, as refinement.
 * @param res String, holds the name of operation,used if no related operation
 * @param signal uml.Signal, helper variable of type uml.Signal
 * @param relationships String, the name of related operation
 * @return listFindOperation List, the list with output statements as String
 */
       module::findOperation(opName : String){
         var res: String = "#"+opName;
         var signal : uml.Signal;
         var relationships : String;
         var listFindOperation : List;
         uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
             c.ownedOperation->forEach(op : uml.Operation){
                 if(opName.equals(op.name)){
                     relationships = hasRelationships(op.qualifiedName);
                     if(relationships.equals("NoRelationships")){
                         listFindOperation.add(res);
                    }
                    else{
                    listFindOperation.add(mapRelationships(relationships,"MethodCall"));
                }
                 }
             }
         }
         uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
             c.ownedReception->forEach(r : uml.Reception){
                     signal = r.signal;
                 if(opName.equals(signal.name)){
                     relationships = hasRelationships(signal.qualifiedName);
                     if(relationships.equals("NoRelationships")){
                      listFindOperation.add(res);
                 }
                 else{
                    listFindOperation.add(mapRelationships(relationships,"SignalReception"));
                 }
                }
             }
         }
         return listFindOperation;
       }
```

```
/**
 * rule mapStateInvariant produces the output for a StateInvariant.
 * @param listStateInvariant List, list of output elements for StateInvariant
 * @return ret String, the total output for a StateInvariant as String
 */
       transitiontree.StateInvariant::mapStateInvariant(){
              var listStateInvariant : List;
              var ret : String;

              listStateInvariant.add("# StateInvariant:" + self.name + "\tValue: " +
self.value + "\n");
              listStateInvariant.add("eval = '" + self.name+"'\n");
              listStateInvariant.add(updateAttList(parseEnumeration(self.value)));
              listStateInvariant->forEach(n : String){
            ret += n;
        }

              return ret;
       }
/**
 * rule parseEnumeration checks if an expression contains the "::" string,
 * if so the string is altered such that only the substring after "::" and
 * substring before whatever word is before "::" are kept, the expression
 * containing the "::" is eliminated. E.g. "..VideoProtocol::Off..' -> 'Off'
 * @param lower Integer, temporary helper variable
 * @param upper Integer, temporary helper variable
 * @param i Integer, temporary helper variable
 * @param index Integer, index of the string "::" in expression
 * @param enum String, the name of enumeration (word before "::")
 * @param param String, temporary helper variable
 * @return value String, the altered expression as String
 */
       module::parseEnumeration(exp : String){
        var upper : Integer;
        var lower : Integer;
        var i : Integer;
        var index : Integer;
        var enum : String;
        var param : String;
        var value : String;

        value = exp;
        index = value.indexOf("::");
        while(index != -1){
            i=index;
            while(i != 0 and not checkCharList(value.charAt(i))){
                i = i-1;
            }
            lower = i+1;
            i = index+2;
            param = "'";
            while(i < value.size() and not checkCharList(value.charAt(i))){
                    param += value.charAt(i);
                    i = i+1;
            }
            param += "'";
            upper = i;
            enum = value.substring(lower,upper);
            value = value.subStringBefore(enum)+ param + value.subStringAfter(enum);
            index = value.indexOf("::");
       }
       return value;
        }
```

```
/**
 * rule isEnumeration gets the name of a parameter and checks if
 * this name is in the list of Enumerations derived from the uml
 * model.Returns true if so, else false.
 * @param name String, the name of the parameter
 * @return found Boolean, the Boolean value of the output as String
*/
	module::isEnumeration(name : String){
		var found : Boolean = false;

		enumerationList->forEach(e:uml.Enumeration){
			if(name.equals(e.name)){
				found = true;
			}
		}
		return found;
	}

/**
 * rule getEnumeration gets the name of a Parameter and calls
 * the java method getRandomInt with upperLimit as seed to get
 * a random value for the index which is used to pick a Enumeration-
 * Literal from the list of owned EnumerationLiterals.Randomly
 * selects an EnumerationLiteral from an Enumeration.
 * @param index Integer, the randomly generated index
 * @param temp Integer, temporary helper variable
 * @param enumerationLiteralList List, the list of Enumeration Literals
 * @return param String, the selected enumeration literal as String
*/
	module::getEnumeration(name : String){
	var index : Integer;
	var temp: Integer = 0;
	var param : String = "undefined";
	var enumerationLiteralList : List;

	enumerationList->forEach(e:uml.Enumeration){
		if(name.equals(e.name)){
			e.ownedLiteral->forEach(l:uml.EnumerationLiteral){
				enumerationLiteralList.add(l.name);
			}
			index = enumerationLiteralList.size();
			index = java ("RandomIntGenerator","getRandomInt",index,javaPath);
			temp = 0;
			enumerationLiteralList->forEach(l : String){
				if(temp==index){
					param = l;
					break;
				}
				temp+=1;
			}
		}
	}
	return param;
	}
```

```
/**
 * rule hasRelationships computes the related operation of a trigger. The operation
 * specified by a trigger can have a refined operation specified and this name
 * is returned, otherwise the string "NoRelationships" is returned. The refined
 * operation will alwas be implemented by the TNG-class in this version of the tool.
 * @param triggerName String, the name of the operation specified in the trigger
 * @return rel String, the name of the related/refined operation as String
 */
       module::hasRelationships(triggerName : String){  //check if Trigger has relationships
             var rel : String = "NoRelationships";
             uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
                 if(c.name.equals("TNG")){
                   c.ownedOperation->forEach(op : uml.Operation){
                     op.clientDependency->forEach(d : uml.Dependency){
                         d.relatedElement->forEach(e){
                            if(e.oclGetType().equals("Operation")){
                              var relOp : uml.Operation = e;
                              if(relOp.qualifiedName.equals(triggerName)){
                                rel = op.qualifiedName;
                                break;
                               }
                              }
                              if(e.oclGetType().equals("Reception")){
                                 var relOp : uml.Reception = e;
                                 if(relOp.signal.qualifiedName.equals(triggerName)){
                                    rel = op.qualifiedName;
                                    break;
                                 }
                              }
                           }
                           }
                     }
                 }
             }
       return rel;
       }

/**
 * rule mapRelationships generates output for each operation that is
 * releted to some other operation. The rule generates basically similar
 * output as the mapTrigger rule, except that this rule is more specific.
 * Since some operations may have related operation and some do not, I needed
 * to have both rules even though in our scenarios all MethodCalls and
 * SignalReceptions are related to a function in the TNG-class.
 * @param name String, the name of the trigger
 * @param type String, the type of the trigger
 * @param listRelationships List, list of generated output statements
 * @param param String, temporary helper variable
 * @param temp String, temporary helper variable
 * @param att String, temporary helper variable
 * @param generatedInteger Integer, the randomly generated integer
 * @return ret String, the final output generated in this rule as String
 */
       module::mapRelationships(name : String, type : String){
         var listRelationship : List;
         var param : String;
         var temp : String;
         var ret : String;
         var att : String;
         var generatedInteger : Integer;

         uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
             if(c.name.equals("TNG")){
            c.ownedOperation->forEach(op : uml.Operation){
                if(op.qualifiedName.equals(name)){     //find related Operation
                    if(type.equals("MethodCall")){
                        param += "\n%>\n\ttesttarget." + op.name+ " ";
                    }
                    if(type.equals("SignalReception")){
                        temp = "$caller";
                        att = "init_var ('caller',";
                        param += att + jClass + "getCaller(\"" + op.name + "\"))";
                    }
```

```
                if(not op.ownedParameter.isEmpty()){
                    op.ownedParameter->forEach(p : uml.Parameter){
                      if(isEnumeration(p.type.name)){
                         param += " " + getEnumeration(p.type.name) + " ";
                         checkGuard(p.name,"caller");
                      }
                      if(p.type.name.equals("Integer")){
                      generatedInteger = java
                  ("RandomIntGenerator","getRandomInt",3,javaPath);
                            param += " " + generatedInteger;
                            checkGuard(p.name,generatedInteger);
                      }
                      if(p.type.name.equals("String")){
                        if(type.equals("MethodCall")){
                            checkGuard(p.name,"caller");
                           temp = "init_var ('caller'," +
                         p.getDerivedOperation(type,"caller")+")";
                            param = temp +  param + "$caller";

                        }
                        else{
                            checkGuard(p.name,"testtarget");
                            param += "\n%>\n";
                            param += "\t" + temp + "." + op.name;
                            param += p.getDerivedOperation(type,"testtarget");
                        }
                      }
                    }
                }
                listRelationship.add("init_var('tng_results',results)");
                listRelationship.add(param);
                listRelationship.add("\twait 5");
                listRelationship.add("\n<%");
                listRelationship.add("try:\n\timport Get_Val\nexcept:");
                listRelationship.add("\tprint os.path.basename(os.path.abspath(__file__))");
                listRelationship.add("\timport Get_Val");
                listRelationship.add("attTable = get_var('tng_attTable')");
                listRelationship.add("t = get_var('tng_t')");
                listRelationship.add("c = get_var('tng_c')");
                listRelationship.add("results = get_var('tng_results')");
                listRelationship.add(getHelperLists());
                listRelationship.add("init_var('tng_attTable',attTable)");
                listRelationship.add("init_var('tng_t',t)");
                listRelationship.add("init_var('tng_c',c)");
            }
        }
           }
        }

    if(not listRelationship.isEmpty()){
       listRelationship->forEach(t : String){
           ret += t + "\n";
       }
    }
    return ret.trim();
    }
```

```
/**
 * rule getDerivedOperation checks if a parameter has a specified
 * derived operation and returns either the string ("testtarget")
 * if type is MethodCall otherwise just the parameter name. This
 * is necessary as some values of parameters,as the caller parameter
 * of the operation dial() needs to be randomly selected.
 * @param type String, the type of the parameter
 * @param att String, the name of the parameter
 * @return derivedOp String, the return statement (tessttarget/att)
 */
 uml.Parameter::getDerivedOperation(type : String,att : String){
    var derivedOp : String = self.name;

        uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
          if(c.name.equals("TNG")){
         c.ownedOperation->forEach(op : uml.Operation){
             op.ownedParameter->forEach(p:uml.Parameter){
                 if(p.qualifiedName.equals(self.qualifiedName)){
                     p.clientDependency->forEach(d : uml.Abstraction){
                         d.relatedElement->forEach(cl){
                             if(cl.oclGetType().equals("Operation")){
                                 var relOp : uml.Operation = cl;
                                 if(relOp.owner.name.equals("TestData")){
                                     if(type.equals("MethodCall")){
                                      derivedOp = " " + jClass + relOp.name +
                                      "(\"testtarget\")";
                                     }
                                     if(type.equals("SignalReception")){
                                         derivedOp = " "  + att;
                                     }
                                 }
                             }
                         }
                     }
                 }
             }
          }
          }
          return derivedOp;
        }

/**
 * rule createAttTable generates the code that instatiates the attribute-
 * table needed in each script. This table is part of the testdata generated
 * in each script. The table holds each attribute from the uml class diagram,
 * along with its properties such as the owning class, object,path to attribute
 * in TNG and so on. The table is looked into as we work with ocl-expressions to
 * find the path to an attribute and to get its class/object/value.
 * @param param String, temporary helper variable
 * @param i Integer, temporary helper variable
 * @param indexOfPath Integer, the index on which the path expression starts
 * @param indexOfSubPath Integer, the index on which the sub-path expression starts
 * @param mult Integer, holds the multiplicity of a class
 * @param temp String, temporary helper variable
 * @param path String, the path to attribute (from class diagram)
 * @param subPath String, the sub-path of attribute (from class diagram)
 * @return ret String, the final generated code for the attribute table as String
 */
        module::createAttTable(){
          var param : String;
          var ret : List;
          var i : Integer = 0;
          var indexOfPath : Integer= 0;
          var indexOfSubpath : Integer = 0;
          var mult : Integer;
          var temp : String;
          var path : String = " ";
          var subPath : String = "[]";

          ret.add("attTable = []");
          uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
         mult = checkMultiplicity(c.name);
         i = 0;
```

```
        if(mult=1){
          c.ownedAttribute->forEach(a : uml.Property){
            if(a.association==null){
                indexOfPath = a.ownedComment.first().body.indexOf("PATH:")+5;
                indexOfSubpath = a.ownedComment.first().body.indexOf("SUBPATH:")-1;
                if(indexOfPath!=-1){
                  temp = a.ownedComment.first().body;
                  if(temp.size() > 0){
                    path = temp.substring(indexOfPath,indexOfSubpath);
                    indexOfSubpath += 9;
                    subPath =  temp.substring(indexOfSubpath,temp.size());
                  }
                }
                param = "attTable.extend(['" + c.name + "','" + c.name.toLower() + "','" +
getRoleName(c.name) +  "','" + a.name +
                "','"+ a.type.name +  "','" + mult + "','" + path.trim() + "'," +
subPath.trim() + "])";
                ret.add(param);
                path = " ";
              }
           }
        }
        else{
            while(i < mult){
            i+=1;
            c.ownedAttribute->forEach(a : uml.Property){
            if(a.association==null){
                indexOfPath = a.ownedComment.first().body.indexOf("PATH:")+5;
                indexOfSubpath = a.ownedComment.first().body.indexOf("SUBPATH:")-1;
                if(indexOfPath != -1){
                  temp = a.ownedComment.first().body;
                  if(temp.size() > 0){
                    path = temp.substring(indexOfPath,indexOfSubpath);
                    indexOfSubpath += 9;
                    subPath =  temp.substring(indexOfSubpath,temp.size());
                  }

                }
                param = "attTable.extend(['" + c.name + "','" +c.name.toLower()+ (i-1)+ "','"+
getRoleName(c.name)+"','" + a.name +  "','"
                + a.type.name +  "','" + mult + "','" + path.trim() + "'," + subPath.trim() +
"])";
                ret.add(param);
                path = " ";
             }
            }
          }
        }
      }
      ret.add(getHelperLists());
      return ret;
       }

/**
 * rule getHelperLists generates the code to instatiate the other
 * smaller lists that are derived from the attibute list generated
 * in rule createAttTable. There is a list for the classes,objects,
 * names,multiplicities,paths,roles and subpaths. Having these lists
 * makes it easier to work with variables in the test-scripts.
 * @return ret String, the code to instatiate the helper lists as String
 */
       module::getHelperLists(){
         var ret: String;
         ret = "\n# Initialize helper lists\n"
         ret += "attClass = attTable[0::8]\n";
      ret += "attNames = attTable[3::8]\n";
      ret += "attObject = attTable[1::8]\n";
      ret += "attPath = attTable[6::8]\n";
      ret += "attRole = attTable[2::8]\n";
      ret += "attMult = attTable[5::8]\n";
      ret += "attSubpath = attTable[7::8]\n";
      ret += "attType = attTable[4::8]\n";
      return ret;
       }
```

```
/**
 * rule getRoleName computes the name of the role for a class
 * given there exsists some association from this class to some
 * other class in the uml class diagram.
 * @param className String, the name of the class
 * @return ret String, the derived role-name as String
 */
       module::getRoleName(className : String){
       var ret : String;
       uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
         c.ownedAttribute->forEach(a : uml.Property){
           if(a.association != null){
             if(className.equals(a.association.endType.first().name)){
                 ret = a.association.memberEnd.first().name;
                 break;
             }
           }
         }
       }
       return ret;
       }
/**
 * rule updateAttList generates the code that checks if an attribute
 * name is contained in an expression and also calls the java-method
 * that evaluates the OCL-expression. The expression can be either
 * a guard on a transition or a State Invariant.
 * @param value String, the expression to inspect
 * @return ret String, the produced code as String
 */
       module::updateAttList(value : String){
         var ret: String;

      ret = "# Start Temporary Consistency \n"
         ret += "existAtt = \"\"\n";
         ret += "value = []\n";
         ret += "j = 0\n";
         ret += "# force reload of status document\n"
      ret += "target = get_system(\"testtarget\")\n";
      ret += "target.status.forcetext\n";
         ret += "for index,name in enumerate(attNames):\n";
         ret += "\texpression = attRole[index]+'.'+name\n";
         ret += "\tif int(attMult[index]) > 1:\n";
         ret += "\t\tvalue =
Get_Val.new_get_value(target,attPath[index],attSubpath[index])\n";
      ret += "\telse:\n\t\tvalue = Get_Val.new_get_value(target,attPath[index])\n";
         ret += "\tif len(value)-1 == 0:\n\t\tj = 0\n";
         ret += "\tif expression in \"" + value + "\":\n";
         ret += "\t\texistAtt += \" \" + attClass[index]\n";
         ret += "\t\texistAtt += \" \" + name\n";
         ret += "\t\texistAtt += \" \" + attObject[index]\n";
         ret += "\t\texistAtt += \" \" + attType[index]\n";
         ret += "\t\ttheValue = \" '\"  + value[j] + \"'\"\n";
         ret += "\t\tif attType[index] == 'Integer':\n";
         ret += "\t\t\ttheValue = \" \" + value[j]\n";
         ret += "\t\texistAtt += theValue\n";
         ret += "\tif j < (len(value)-1):\n\t\tj = j+1\n";
         ret += "print \"ExistAtt: \",existAtt\n";
         ret += "queryResult = c.updateObjectDiagram(existAtt,\"" + value.trim() + "\")\n";
         ret += "print \"Result of query: \", queryResult\n";
         ret += "results.extend([eval,queryResult])\n";
         ret += "# Finisehed Temporary Consistency\n";
         return ret.trim();
       }
```

```
/**
 * rule checkMultiplicity computes the multiplicity of a class specified
 * in the uml class diagram.
 * @param class String, the name of the class to compute multiplicity for
 * @return mult Integer, the computed multiplicity
 */
       module::checkMultiplicity(class : String){
         var mult : Integer = 1;
         uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
           if(not c.name.equals(class)){
             c.ownedAttribute->forEach(a : uml.Property){
             if(a.association!=null){
               if(class.equals(a.association.memberEnd.first().type.name)){
                 mult = a.association.memberEnd.first().upper *
checkMultiplicity(a.association.memberEnd.last().type.name);
               }
             }
             }
           }
         }
         return mult;
       }
/**
 * rule getImportStatements generates the code that on top of each
 * produced test-script. The code mainly consists of import statements
 * and there is an import statements for the Get_Val module. This module
 * is written in Python and has functions that combined compute the
 * current value of an attribute on run-time, given the attributes
 * path and subpath amon other things.
 * @param statements List, the list of generated output elements
 * @param ret String, the final output as String
 */
       module::getImportStatements(){
              var statements : List;
              var ret : String
              statements.add("<%\nfrom jpype import *");
              statements.add("import os.path");
              statements.add("import jpype");
              statements.add("sys.path.append(os.path.basename(os.path.abspath(__file__)))");
        statements.add("sys.path.append('.')");
        statements.add("try:\n\timport Get_Val\nexcept:");
        statements.add("\tprint os.path.basename(os.path.abspath(__file__))");
        statements.add("\timport Get_Val");
              statements.addAll(createAttTable());
              statements.add("classpath = os.path.join(os.path.abspath('.'))");
              statements.add("jpype.startJVM(\"jvm.dll\", \"-Djava.ext.dirs=%s\"
              %classpath)");
              statements.add("Test = jpype.JClass('Test.TestData')");
              statements.add("ClassDiagram = jpype.JClass('Test.ClassDiagramTestData')");
              statements.add("t = Test()");
              statements.add("c = ClassDiagram()");
              statements.add("results = []");
              statements.add("init_var('tng_t',t)");
              statements.add("init_var('tng_c',c)");
              statements.add("init_var('tng_attTable',attTable)\n");
              statements->forEach(s : String){     //format output
                     ret += s + "\n";
              }
              return ret.trim();
       }
}
```

9.8 Appendix H – ClassDiagramTestdata.m2t (MOFScript)

```
/**
 * @author Miran Damjanovic
 * @date 12.03.2009
 * This m-2-t transformation generates the test-data need by the
 * TNG-scripts. The purpose of the generated java-class is to
 * implement a class-diagram and an object-diagram in order to
 * be able to use the IEOS java library.The class- and object-
 * diagram are derived from the domain model (class diagram), which
 * is also the input model to this transformation.
 */


/**
 * Text-transformation ClassDiagramTestData. Has a main rule that invokes
 * the other rules of the transformation to build up the file, creates
 * an enumerationlist and also prints the file itself in the end.
 * @param uml in, the input model (class diagram)
 * @param enumerationList List, list of enumerations derived from the input model
 * @param queryContext String, context of the query to be executed by scripts
 */
texttransformation ClassDiagramTestData(in uml:"http://www.eclipse.org/uml2/2.0.0/UML" ) {
  var enumerationList : List;
  var queryContext : String = "context Saturn inv name: ";

/**
 * rule main() starts the transformation process
 * @param fileList List, list of elements to be printed to file
 * @param ret String, temoprary helper variable
 * @param f file, the generated output file
 */
  module::main(){
    var fileList : List;
    var ret : String;

    uml.objectsOfType(uml.Enumeration)->forEach(e : uml.Enumeration){
        enumerationList.add(e);
    }

    fileList.add(getImportStatements());
    fileList.add("\t\tieos.createClassDiagram();\n");
    uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
      fileList.add("\t\tieos.insertClass(\"" + c.name + "\");");
    }
    uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
        ret = c.mapClass();
        if(not ret.equals("")){
            fileList.add(ret);
        }
    }
    fileList.add("\t\tieos.closeClassDiagram();\n");
    fileList.add("\t\tieos.createObjectDiagram();\n");
    fileList.addAll(instantiateObjectDiagram());
    fileList.add(getCodeForUpdatingObjects());
    fileList.add("\t}\n");
    fileList.add("}\n");
    file f("ClassDiagramTestData.java");
    fileList->forEach(c){
            f.println(c);
    }
  }
```

```
/**
 * rule mapClass generates the appropriate output for each class
 * from the input class diagram according to the IEOS.
 * @param listClass List, the list of generated outputs
 * @param temp String, temporary helper variable
 * @return ret String, the generated class diagram output
 */
 uml.Class::mapClass(){
   var listClass : List;
   var ret : String;
   var temp : String;

   if(not self.ownedAttribute.isEmpty()){
       self.ownedAttribute->forEach(a : uml.Property){
           if(a.association!=null){
             temp = "ieos.insertAssociation(";
             temp += "\"" + a.association.memberEnd.last().type.name + "\",";
             temp += "\"" + a.association.memberEnd.last().name + "\",";
             if(a.association.memberEnd.last().upper==-1){
               temp += "\"*\","
             }
             if(a.association.memberEnd.last().upper >=
                       a.association.memberEnd.last().lower){
                        temp += "\"" + a.association.memberEnd.last().lower + ".." +
                        a.association.memberEnd.last().upper + "\",";
             }
             if(a.association.memberEnd.first().upper==-1){
               temp += "\"*\","
             }
             if(a.association.memberEnd.first().upper >=
                       a.association.memberEnd.first().lower){
                        temp += "\"" + a.association.memberEnd.first().lower + ".." +
                        a.association.memberEnd.first().upper + "\",";
             }
             temp += "\"" + a.association.memberEnd.first().name + "\",";
             temp += "\"" + a.association.memberEnd.first().type.name + "\");";
             listClass.add(temp);
           }
           else{
               if(not isEnumeration(a.type.name)){
                   listClass.add("ieos.insertAttribute(\"" + self.name + "\"," +
                   " \"" + a.name + "\", " + "\"" + a.type.name + "\");");
               }
               else{
                   listClass.add("ieos.insertAttribute(\"" + self.name + "\"," +
                   " \"" + a.name + "\", " + "\"String\");");
               }
           }
       }
   }
   if(not listClass.isEmpty()){
       listClass->forEach(t : String){
           ret += "\t\t" + t + "\n";
       }
   }
   return ret;
 }

/**
 * rule isEnumeration checks if an attribute type is an
 * Enumeration. Returns true if it is, false otherwise.
 * @return found Boolean, the computed Boolean value
 */
 module::isEnumeration(name : String){
       var found : Boolean = false;
       enumerationList->forEach(e:uml.Enumeration){
           if(name.equals(e.name)){
               found = true;
           }
       }
       return found;   //return statement
   }
```

```
/**
 * rule instatiateObjectDiagram generates the code for creating
 * the object diagram as specified in the IEOS.
 * @param helperList HashTable, list of the first memebers in associations
 * @param param String, temporary helper variable
 * @param default String, default value inserted in object diagram (parameter)
 * @param passed Boolean, temporary helper variable
 * @param i Integer, temporary helper variable
 * @param j Integer, temporary helper variable
 * @param multSource Integer, multiplicity of association source
 * @param multTarget Integer, multiplicity of association target
 */
  module::instantiateObjectDiagram(){
    var ret : List;
    var helperList : Hashtable;
    var param : String;
    var i : Integer = 0;
    var j : Integer = 0;
    var multSource : Integer;
    var multTarget : Integer;
    var default : String;
    var passed : Boolean = false;

    uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
        multSource = checkMultiplicity(c.name);
        if(multSource == 1){

          param = "\t\tieos.insertObject(\"" + c.name + "\",\"" + c.name.toLower() + "\");";
          ret.add(param);
        }
        else{
           i=0;
            while(i<multSource){
            i += 1;
            param = "\t\tieos.insertObject(\"" + c.name + "\",\"" + c.name.toLower() +(i-
1)+"\");";
            ret.add(param);
            }
        }
    }

    uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
        multSource = checkMultiplicity(c.name);
        if(multSource==1){
            c.ownedAttribute->forEach(a : uml.Property){
            passed = false;
            if(a.association==null){
                default = "Off";
                if(a.type.name.equals("Integer")){
                  default = "0";
                }
                param = "\t\tieos.insertValue(\"" + c.name + "\",\"" + a.name + "\",\"" +
                        c.name.toLower() +  "\",\"" + default + "\");";
                ret.add(param);
            }
            else{
                if(a.association.memberEnd.first().upper < 2){
                  param = "\t\tieos.insertLink(\"" + a.association.memberEnd.last().type.name
+ "\",\"" +
               a.association.memberEnd.last().type.name.toLower()+ "\",\"" +
               a.association.memberEnd.last().name + "\",\"" +
               a.association.memberEnd.first().name + "\",\"" +
               a.association.memberEnd.first().type.name.toLower() +  "\",\"" +
                                    a.association.memberEnd.first().type.name + "\");";
                                    ret.add(param);
                }
```

```
            else{
                  j = 0;
                  while(j < a.association.memberEnd.first().upper){
                       j += 1;
                     param = "\t\tieos.insertLink(\"" +
                  a.association.memberEnd.last().type.name + "\",\"" +
                  a.association.memberEnd.last().type.name.toLower()+ "\",\"" +
                  a.association.memberEnd.last().name + "\",\"" +
                  a.association.memberEnd.first().name
                                    + "\",\"" +
                  a.association.memberEnd.first().type.name.toLower() + (j-1)+ "\",\"" +
                  a.association.memberEnd.first().type.name + "\");";
                  ret.add(param);
                  }
                 }
               }
            }
         }
         else{
              i=0;

              while(i<multSource){
              i += 1;
              c.ownedAttribute->forEach(a : uml.Property){
                  if(a.association==null){
                      default = "Off";
                      if(a.type.name.equals("Integer")){
                          default = "0";
                      }
                      param = "\t\tieos.insertValue(\"" + c.name + "\",\"" + a.name + "\",\"" +
                              c.name.toLower() + (i-1) + "\",\"" + default + "\");";
                      ret.add(param);
                  }
                  else{
                      j = 0;
                      while(j < a.association.memberEnd.first().upper){
                        j += 1
                        if(!passed){
                        multTarget = checkMultiplicity(
                        a.association.memberEnd.first().type.name);
                          helperList = getHelperList();
                          passed = true;
                        }
                       multTarget = helperList.get(a.association.memberEnd.first().type.name)-1;
                       helperList.put(a.association.memberEnd.first().type.name,multTarget)
                        param = "\t\tieos.insertLink(\"" +
                        a.association.memberEnd.last().type.name + "\",\"" +
                        a.association.memberEnd.last().type.name.toLower()+(i-1)+ "\",\"" +
                        a.association.memberEnd.last().name + "\",\"" +
                        a.association.memberEnd.first().name
                        + "\",\"" + a.association.memberEnd.first().type.name.toLower() +
                        multTarget + "\",\"" +
                        a.association.memberEnd.first().type.name + "\");";
                        multTarget = multTarget - 1;
                        ret.add(param);
                     }
                  }
                }
              }
           }
        }
    }
    return ret
}
```

```
/**
 * rule checkMultiplicity computes the multiplicity of a class.
 * @param class String, the name of the class
 * @return mult Integer, computed multiplicity of the class
 */
  module::checkMultiplicity(class : String){
      var mult : Integer = 1;
      uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
        if(not c.name.equals(class)){
          c.ownedAttribute->forEach(a : uml.Property){
              if(a.association!=null){
                if(class.equals(a.association.memberEnd.first().type.name)){
                 mult = a.association.memberEnd.first().upper*
       checkMultiplicity(a.association.memberEnd.last().type.name);
                }
              }
          }
        }
      }
      return mult;
    }

/**
 * rule getCodeForUpdatingObjects generates the code necessary to
 * execute a query in IEOS. This code is placed at the bottom of
 * the generated class (ClasssDiagramTestData).
 * @return ret String, the generated code
 */
   module::getCodeForUpdatingObjects(){
    var ret : String;
    ret = "\n\t\tString queryResult = null;\n";
    ret += "\t\tString phrase = updateData;\n";
    ret += "\t\tString delims = \" \";\n";
    ret += "\t\tString[] tokens = phrase.split(delims);\n";
    ret += "\t\tif(tokens.length > 5){\n";
    ret += "\t\t\tfor (int i = 1; i < tokens.length; i+=5){\n";
    ret += "\t\t\t\tString value = tokens[i+4];\n";
    ret += "\t\t\t\tString type = tokens[i+3];\n";
    ret += "\t\t\t\tif(value.equals(\"undefined\")){\n";
    ret += "\t\t\t\t\tvalue = \"Off\";\n";
    ret += "\t\t\t\t\tif(type.equals(\"Integer\")){\n";
    ret += "\t\t\t\t\t\tvalue = \"0\";\n\t\t\t\t\t}\n";
    ret += "\t\t\t\t\tif(type.equals(\"String\")){\n";
    ret += "\t\t\t\t\t\tvalue = \" \";\n\t\t\t\t\t}\n\t\t\t\t}\n";
    ret += "\t\tSystem.out.println(\"(UPDATING) Class: \" + tokens[i] + \",Att: \" +
tokens[i+1] + \",Obj: \" + tokens[i+2] +\",UpdateValue: \" + value);\n";
    ret += "\t\tieos.insertValue(tokens[i],tokens[i+1],tokens[i+2],value);\n";
    ret += "\t\t}\n\t\t}\n";
    ret += "\t\ttry{\n";
    ret += "\t\tSystem.out.println(\"Closing ObjectDiagram..\");\n";
    ret += "\t\tieos.closeObjectDiagram();\n\t\t}\n";
    ret += "\tcatch (Exception e) {\n";
    ret += "\t\tSystem.out.println(\"Unable to close ObjectDiagram..\");\n";
    ret += "\t\te.printStackTrace();\n\t\t}\n";
    ret += "\tString query = \"" + queryContext + "\" + queryValue;\n";
    ret += "\ttry{\n";
    ret += "\t\tSystem.out.println(\"Executing Query: \" + query);\n";
    ret += "\t\tqueryResult = ieos.query(query);\n\t\t}\n";
    ret += "\tcatch (Exception e) {\n";
    ret += "\t\te.printStackTrace();\n";
    ret += "\t\tSystem.out.println(\"Query failed..\");\n\t\t}\n";
    ret += "\treturn queryResult;\n";

    return ret;
   }
```

```
/**
 * rule getHelperList creates an HashTable with names of classes that are
 * targets of associatins in the class diagram and their multiplicities.
 * @return ret HashTable, the computed HashTable
 */
  module::getHelperList(){
  var ret : Hashtable;
  uml.objectsOfType(uml.Class)->forEach(c : uml.Class){
    c.attribute->forEach(a : uml.Property){
        if(a.association != null){

ret.put(a.association.memberEnd.first().type.name,checkMultiplicity(a.association.memberEnd.fi
rst().type.name));
        }
    }
  }
  return ret;
 }

/**
 * rule getImportStatements generates the initial statements.
 * @return ret String, the generated output
 */
  module::getImportStatements(){
    var ret : String;
    ret = "package Test;\n";
    ret += "import core.IEOS; \n";
    ret += "import java.util.*;\n";
    ret += "public class ClassDiagramTestData { \n";
    ret += "\tprivate IEOS ieos; \n\n";
    ret += "\tpublic ClassDiagramTestData(){ \n";
    ret += "\t}\n";
    ret += "\tpublic String updateObjectDiagram(String updateData, String queryValue) throws
Exception{\n";
    ret += "\t\tieos = new IEOS();\n";
    return ret;
  }
}
```

9.9 Appendix I – ClassDiagramTestData.java (Java)

```
package Test;
import core.IEOS;
import java.util.*;
public class ClassDiagramTestData {
	private IEOS ieos;

	public ClassDiagramTestData(){
	}
	public String updateObjectDiagram(String updateData, String queryValue) throws
Exception{
	ieos = new IEOS();
	ieos.createClassDiagram();
	ieos.insertClass("Saturn");
	ieos.insertClass("TNG");
	ieos.insertClass("TestData");
	ieos.insertClass("SystemUnit");
	ieos.insertClass("Conference");
	ieos.insertClass("Call");
	ieos.insertClass("IncomingVideoChannel");
	ieos.insertClass("OutgoingVideoChannel");
	ieos.insertClass("StateMachine1");
	ieos.insertAssociation("Saturn","OWNER_SATURN","*","0..1","systemUnit","SystemUnit");
	ieos.insertAssociation("Saturn","","*","0..1","conference","Conference");
	ieos.insertAttribute("SystemUnit", "NumberOfActiveCalls", "Integer");
	ieos.insertAttribute("SystemUnit", "MaximumNumberOfCalls", "Integer");
	ieos.insertAttribute("Conference", "PresentationMode", "String");
	ieos.insertAssociation("Conference","","0..1","0..3","calls","Call");
	ieos.insertAttribute("Call", "CallRate", "Integer");
	ieos.insertAssociation("Call","","*","0..2","incomingVideoChannel","IncomingVideoChanne
l");
	ieos.insertAssociation("Call","","*","0..2","outgoingVideoChannel","OutgoingVideoChanne
l");
	ieos.insertAttribute("IncomingVideoChannel", "Protocol", "String");
	ieos.insertAttribute("OutgoingVideoChannel", "Protocol", "String");
	ieos.closeClassDiagram();
	ieos.createObjectDiagram();
	ieos.insertObject("Saturn","saturn");
	ieos.insertObject("TNG","tng");
	ieos.insertObject("TestData","testdata");
	ieos.insertObject("SystemUnit","systemunit");
	ieos.insertObject("Conference","conference");
	ieos.insertObject("Call","call0");
	ieos.insertObject("Call","call1");
	ieos.insertObject("Call","call2");
	ieos.insertObject("IncomingVideoChannel","incomingvideochannel0");
	ieos.insertObject("IncomingVideoChannel","incomingvideochannel1");
	ieos.insertObject("IncomingVideoChannel","incomingvideochannel2");
	ieos.insertObject("IncomingVideoChannel","incomingvideochannel3");
	ieos.insertObject("IncomingVideoChannel","incomingvideochannel4");
	ieos.insertObject("IncomingVideoChannel","incomingvideochannel5");
	ieos.insertObject("OutgoingVideoChannel","outgoingvideochannel0");
	ieos.insertObject("OutgoingVideoChannel","outgoingvideochannel1");
	ieos.insertObject("OutgoingVideoChannel","outgoingvideochannel2");
	ieos.insertObject("OutgoingVideoChannel","outgoingvideochannel3");
	ieos.insertObject("OutgoingVideoChannel","outgoingvideochannel4");
	ieos.insertObject("OutgoingVideoChannel","outgoingvideochannel5");
	ieos.insertObject("StateMachine1","statemachine1");
	ieos.insertLink("Saturn","saturn","OWNER_SATURN","systemUnit","systemunit","SystemUnit"
);
	ieos.insertLink("Saturn","saturn","","conference","conference","Conference");
	ieos.insertValue("SystemUnit","NumberOfActiveCalls","systemunit","2");
	ieos.insertValue("SystemUnit","MaximumNumberOfCalls","systemunit","0");
	ieos.insertValue("Conference","PresentationMode","conference","Off");
	ieos.insertLink("Conference","conference","","calls","call0","Call");
	ieos.insertLink("Conference","conference","","calls","call1","Call");
	ieos.insertLink("Conference","conference","","calls","call2","Call");
	ieos.insertValue("Call","CallRate","call0","0");
	ieos.insertLink("Call","call0","","incomingVideoChannel","incomingvideochannel5","Incom
ingVideoChannel")
	ieos.insertLink("Call","call0","","incomingVideoChannel","incomingvideochannel4","Incom
ingVideoChannel");
```

```
        ieos.insertLink("Call","call0","","outgoingVideoChannel","outgoingvideochannel5","Outgo
ingVideoChannel");
        ieos.insertLink("Call","call0","","outgoingVideoChannel","outgoingvideochannel4","Outgo
ingVideoChannel");
        ieos.insertValue("Call","CallRate","call1","0");
        ieos.insertLink("Call","call1","","incomingVideoChannel","incomingvideochannel3","Incom
ingVideoChannel");
        ieos.insertLink("Call","call1","","incomingVideoChannel","incomingvideochannel2","Incom
ingVideoChannel");
        ieos.insertLink("Call","call1","","outgoingVideoChannel","outgoingvideochannel3","Outgo
ingVideoChannel");
        ieos.insertLink("Call","call1","","outgoingVideoChannel","outgoingvideochannel2","Outgo
ingVideoChannel");
                ieos.insertValue("Call","CallRate","call2","0");
        ieos.insertLink("Call","call2","","incomingVideoChannel","incomingvideochannel1","Incom
ingVideoChannel");
        ieos.insertLink("Call","call2","","incomingVideoChannel","incomingvideochannel0","Incom
ingVideoChannel");
        ieos.insertLink("Call","call2","","outgoingVideoChannel","outgoingvideochannel1","Outgo
ingVideoChannel");
        ieos.insertLink("Call","call2","","outgoingVideoChannel","outgoingvideochannel0","Outgo
ingVideoChannel");
        ieos.insertValue("IncomingVideoChannel","Protocol","incomingvideochannel0","Off");
        ieos.insertValue("IncomingVideoChannel","Protocol","incomingvideochannel1","Off");
        ieos.insertValue("IncomingVideoChannel","Protocol","incomingvideochannel2","Off");
        ieos.insertValue("IncomingVideoChannel","Protocol","incomingvideochannel3","Off");
        ieos.insertValue("IncomingVideoChannel","Protocol","incomingvideochannel4","Off")
        ieos.insertValue("IncomingVideoChannel","Protocol","incomingvideochannel5","Off");
        ieos.insertValue("OutgoingVideoChannel","Protocol","outgoingvideochannel0","Off");
        ieos.insertValue("OutgoingVideoChannel","Protocol","outgoingvideochannel1","Off");
        ieos.insertValue("OutgoingVideoChannel","Protocol","outgoingvideochannel2","Off");
        ieos.insertValue("OutgoingVideoChannel","Protocol","outgoingvideochannel3","Off");
        ieos.insertValue("OutgoingVideoChannel","Protocol","outgoingvideochannel4","Off");
        ieos.insertValue("OutgoingVideoChannel","Protocol","outgoingvideochannel5","Off");
                String queryResult = null;
                String phrase = updateData;
                String delims = " ";
                String[] tokens = phrase.split(delims);
                if(tokens.length > 5){
                        for (int i = 1; i < tokens.length; i+=5){
                                String value = tokens[i+4];
                                String type = tokens[i+3];
                                if(value.equals("undefined")){
                                        value = "Off";
                                        if(type.equals("Integer")){
                                                value = "0";
                                        }
                                        if(type.equals("String")){
                                                value = " ";
                                        }
                                }
                System.out.println("(UPDATING) Class: " + tokens[i] + ",Att: " + tokens[i+1] +
                ",Obj: " + tokens[i+2] +",UpdateValue: " + value);
                                ieos.insertValue(tokens[i],tokens[i+1],tokens[i+2],value);
                                }
                }
                try{
                        System.out.println("Closing ObjectDiagram..");
                        ieos.closeObjectDiagram();
                }
                catch (Exception e) {
                        System.out.println("Unable to close ObjectDiagram..");
                        e.printStackTrace();
                }
                String query = "context Saturn inv name: " + queryValue;
                try{
                        System.out.println("Executing Query: " + query);
                        queryResult = ieos.query(query);
                }
                catch (Exception e) {
                        e.printStackTrace();
                        System.out.println("Query failed..");
                }
                return queryResult;
        }
}
```

9.10 Appendix J – TestData.java (Java)

```
package Test;
import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Vector;

/*
* @author Miran Damjanovic
* @date 22.02.2009
* This class contains methods invoked by the TNG-testscripts at runtime.
* The main idea is to have a vector (array or collection) that indicates
* which of the endpoints are connected or are currently presenting. The
* vector puts a 1 in the index after the endpoint if the endpoint is
* connected/presenting and 0 otherwise. One must also differentiate the
* cases where the caller of the method is an endpoint or the testtarget.
* The names and ip-adresses of the endpoints are picked up from the
* test.tts file.
*/

/*
 * Class TestData is the main class, without the main method. It owns
 * all the methods awailable for invokation at runtime by the test-scripts.
 * @param endPoints Vector, the vector holding the info on the endpoints
 */
public class TestData {
	@SuppressWarnings("unchecked")
	Vector endPoints = new Vector();

	/*
	 * This method reads the tts-file which contains the names and ip-
	 * addresses of the endpoints used in the test, and initializes the
	 * vetor which now will contain endpoint-data.
	 * @param name String, the name of the file
	 * @param dis DataInputStream, used for reading the file
	 * @param line String, temporary helper variable
	 * @param stop int, helper variable indicates the stopping criteria
	 * @return endPoint String, temporary helper variable
	 */
	@SuppressWarnings({ "unchecked", "deprecation" })
	public Vector readFile(String name){
		DataInputStream dis = null;
		String    line = null, endPoint = null;
		int stop = 0;


      try {
        File f = new File(name);
        FileInputStream fis = new FileInputStream(f);
        BufferedInputStream bis = new BufferedInputStream(fis);
        dis = new DataInputStream(bis);
```

```
    while ( (line=dis.readLine()) != null ) {
      stop = line.indexOf('.');
      endPoint = line.substring(0, stop);
      endPoints.add(endPoint);
      endPoints.add(0);
      endPoints.add(0);
  }
}
catch (IOException e) {
  System.out.println("Error : FILE NOT FOUND..");

}
finally {
  //close the file
  if (dis != null) {
        try {
              dis.close();
        }
        catch (IOException ioe) {
              System.out.println("Error : UNABLE TO CLOSE FILE..");
        }
  }
}
return endPoints;
    }

    /*
     * This method just prints out the information held by the vector to
     * stdout in a nice format.
     * @param i int, temporary helper variable used for indexing
     */
    public void printEndPoints(){
            int i = 0;
            while(i < endPoints.size()){
                        System.out.println("Name: " + endPoints.elementAt(i) + ",
                        Connected: " + endPoints.elementAt(i+1)
                        + " , Presenting: " + endPoints.elementAt(i+2));
                        i+=3;
       }
    System.out.println("-----------------------------------------------");
    }
```

```
/*
	 * This method returns a name of a caller  based on the given
	 * command. The command implies a method-call. The method returns an
	 * endpoint that is not connected or presenting currently.
	 * @param command String, the given command-name (method-call)
	 * @param i int, temporary helper variable used for indexing
	 * @param j int, temporary helper variable used for indexing
	* @param presenting boolean, indicates weather the endpoint is *presenting
	 * @return nameOfCaller String, the name of the assigned caller
	 */
	@SuppressWarnings("unchecked")
	public String getCaller(String command){
		String nameOfCaller = "Undefined";
		int i = 0;
		int j = 0;
		boolean presenting = false;

		if(command.equals("dial")){			//command is dial
			while(i < endPoints.size()){		//check if there are available endPoints
				if(endPoints.elementAt(i+1).toString().equals("0")){
					nameOfCaller = endPoints.elementAt(i).toString();
					endPoints.remove(i+1);
					endPoints.add(i+1, 1);
					i = endPoints.size();
				}
				i+=3;
			}
			if(nameOfCaller.equals("Undefined")){
				nameOfCaller = "NoAvailableEndPoints";
			}
		}
		if(command.equals("disconnect")){		//command is disconnect
			while(i < endPoints.size()){		//check if there are connected endPoints
				if(endPoints.elementAt(i+1).toString().equals("1")){
					nameOfCaller = endPoints.elementAt(i).toString();
					endPoints.remove(i+1);
					endPoints.add(i+1, 0);
					i = endPoints.size();
				}
				i+=3;
			}
			if(nameOfCaller.equals("Undefined")){
				nameOfCaller = "NoConnectedEndPoints";
			}
		}

		if(command.equals("openduo")){	//command is openduo
			while(i < endPoints.size()){		//check if allready presenting
				if(endPoints.elementAt(i+2).toString().equals("1")){
					presenting = true;
					j=i;
				}
				i+=3;
			}
			if(presenting){			//allready presenting,close and assign presentation to new
endPoint
				endPoints.remove(j+2);
				endPoints.add(j+2, 0);
				i=0;
				while(i < endPoints.size()){
```

```
                                        if(endPoints.elementAt(i+1).toString().equals("1") && j!=i){
                                                endPoints.remove(i+2);
                                                endPoints.add(i+2, 1);
                                                nameOfCaller = endPoints.elementAt(i).toString();
                                                i=endPoints.size();
                                        }
                                        i+=3;
                                }
                                if(nameOfCaller.equals("Undefined")){
                                        nameOfCaller = "NoConnectedEndPoints";
                                }
                        }
                        else{   //no current presentation,assign presentation to a connected endPoint
                                i=0;
                                while(i < endPoints.size()){
                                        if(endPoints.elementAt(i+1).toString().equals("1")){
                                                endPoints.remove(i+2);
                                                endPoints.add(i+2, 1);
                                                nameOfCaller = endPoints.elementAt(i).toString();
                                                i=endPoints.size();
                                        }
                                        i+=3;
                                }
                                if(nameOfCaller.equals("Undefined")){
                                        nameOfCaller = "NoConnectedEndPoints";
                                }
                        }
                }
                if(command.equals("closeduo")){
                        while(i < endPoints.size()){
                                if(endPoints.elementAt(i+2).toString().equals("1")){
                                        endPoints.remove(i+2);
                                        endPoints.add(i+2, 0);
                                        nameOfCaller = endPoints.elementAt(i).toString();
                                }
                                i+=3;
                        }
                        if(nameOfCaller.equals("Undefined")){
                                nameOfCaller = "NoPresentingEndPoints";
                        }
                }
                System.out.println("Endpoints after getCaller("+command+"):");
                printEndPoints();
                return nameOfCaller;
        }

        /*
         * This method returns a name of an endpoint that another endpoint can
         * connect to. We distinguish between testtarget and other endpoints.
         * @param nameOfCaller String, the name of caller (testtarget or other endpoint)
    * @param i int, temporary helper variable used for indexing
         * @return name String, the assigned number (name of endpoint)
         */
        @SuppressWarnings("unchecked")
        public String getNumberToConnect(String nameOfCaller){
                String name = "Undefined";
                int i = 0;
                if(nameOfCaller.equals("testtarget")){
                        while(i < endPoints.size()){
                                if((endPoints.elementAt(i+1).toString()).equals("0") &&
```

```
!(endPoints.elementAt(i).toString().equals(nameOfCaller))){
                                        name = endPoints.elementAt(i).toString();
                                        endPoints.remove(i+1);
                                        endPoints.add(i+1, 1);
                                        break;
                                }
                                i+=3;
                        }
                        if(name.equals("Undefined")){
                                name = "NoAvailableEndPoints";
                        }
                }
                else{
                        while(i < endPoints.size()){
                                if(endPoints.elementAt(i).toString().equals(nameOfCaller) &&
                                                (endPoints.elementAt(i+1).toString()).equals("0")){
                                        name = "testtarget";
                                        endPoints.remove(i+1);
                                        endPoints.add(i+1, 1);
                                }
                                i+=3;
                        }
                        if(name.equals("Undefined")){
                                name = "AllreadyConnected";
                        }
                }
                System.out.println("Endpoints after getNumberToConnect("+nameOfCaller+"):");
                printEndPoints();
                return name;
        }

        /*
         * This method returns a name of an endpoint that another endpoint can
         * disconnect from. We distinguish between testtarget and other endpoints.
         * @param nameOfCaller String, the name of caller (testtarget or other endpoint)
    * @param i int, temporary helper variable used for indexing
         * @return name String, the assigned number (name of endpoint)
         */
        @SuppressWarnings("unchecked")
        public String getNumberToDisconnect(String nameOfCaller){
                String name = "Undefied";
                int i = 0;
                if(nameOfCaller.equals("testtarget")){
                        while(i < endPoints.size()){
                                if((endPoints.elementAt(i+1).toString()).equals("1") &&
                                        !(endPoints.elementAt(i).toString().equals(nameOfCaller))){
                                        name = endPoints.elementAt(i).toString();
                                        endPoints.remove(i+1);
                                        endPoints.add(i+1, 0);
                                        endPoints.remove(i+2);
                                        endPoints.add(i+2,0);
                                        break;
                                }
                                i+=3;
                        }
                        if(name.equals("Undefined")){
                                name = "NoAvailableEndPoints";
                        }
                }
```

```
		else{
			while(i < endPoints.size()){
				if(endPoints.elementAt(i).toString().equals(nameOfCaller) &&
					(endPoints.elementAt(i+1).toString()).equals("1")){
					name = "testtarget";
					endPoints.remove(i+1);
					endPoints.add(i+1, 0);
					endPoints.remove(i+2);
					endPoints.add(i+2,0);
				}
				i+=3;
			}
			if(name.equals("Undefined")){
				name = "AllreadyDisconnected";
			}
		}
		System.out.println("Endpoints after getNumberToDisconnect("+nameOfCaller+"):");
		printEndPoints();
		return name;
}

/*
 * This method iterates through the vector and creates a string with
 * the values from the vector presented as String. It is used for keeping
 * the vector consistent while running the script, as the test-script is
 * composed of several Python-parts and TNG-parts, the vector has to be
 * set every time we enter a Python part.
 * @param i int, temporary helper variable used for indexing
 * @return ret String, the derived string with endpoint data
 */
public String getEndPoints(){
		String ret = "";
		int i = 0;
		while(i < endPoints.size()){
			ret += endPoints.elementAt(i).toString()+" ";
			i++;
		}
		return ret;
}

/*
 * This method sets the endpoint data in the vector.
 * @param endP String, the string with endpoint data
 * @param delims String, helper variable used for working on Strings
 * @param tokens Array, holds the split tokens (words)
 * @param i int, temporary helper variable used for indexing
 */
@SuppressWarnings("unchecked")
public void setEndPoints(String endP){
		endPoints.clear();
```

```
                String delims = " ";
                String[] tokens = endP.split(delims);
                int i=0;
                while(i < tokens.length){
                        endPoints.add(tokens[i]);
                        i++;
                }
        }

        /*
         * The constructor for the TestData class. It only invokes the readfile()-
         * method that initializes the endpoint-vector.
         */
        public TestData(){
                readFile("test.tts");
        }
}
```

9.11 Appendix K – Get_Val.py (Python)

```
from com.api.cuil.exc import NoSuchElement

def new_get_value(target,path,multiple=None):
    """ target equals the testpoint,
        path is the xstatus path
        multiple if we expect multiple instances
    """
    try:
        values = []
        if multiple == None:
            try:
                val =eval("target.%s.forcetext" % path)
            except NoSuchElement,e:
                val = 'undefined'
            values.append(val)
        else:
            paths = createPaths(target,multiple)
            for createdPath in paths:
                try:
                    values.append(eval("target.%s.%s.forcetext" %
(createdPath,path)))
                except NoSuchElement,e:
                    print "Exception from Get_Val: ",e
                    values.append('undefined')

        return values
    except Exception, e:
        print "Exception from Get_Val: ",e

        return ['undefined']

def createPaths(target,paths):
    """ paths = ['status.MediaChannels.Call','IncomingVideoChannel']
        callId,channelId
    """
    if len(paths) == 0:
        return 'undefined'
    else:
        path       = paths.pop(0)
        elementIds = get_node_ids(target,path)
        return subPath(target,path+'[%s]',paths,elementIds)

def subPath(target,path,paths,elementsId):
    createdPaths = []
    if len(paths) == 0:
        for id in elementsId:
            createdPaths.append(path % id)
        return createdPaths

    # status.MediaChannels.Call[%s].IncomingVideoChannel
    path += ".%s" % paths.pop(0)
    for id in elementsId:
        # status.MediaChannels.Call[2].IncomingVideoChannel
        tmpPath = path % id
        tmpElementId = get_node_ids(target,tmpPath)
        createdPaths.extend(subPath(target,tmpPath+"[%s]",paths,tmpElementId))
    return createdPaths


def get_node_ids(target,paths):
    """ Return the path ids of the existing paths as a list eg:
('187','298,'309')"""
    return eval('target.%s.get_items()' % paths)
```

9.12 Appendix L – RandomIntGenerator.java (Java)

```
import java.util.*;
public class RandomIntGenerator {

	/**
	 * @author Miran Damjanovic
	 * @date 13.03.2009
	 * This class has a function, getrandomInt(), that produces
	 * a random Integer according to the given seed value. This
	 * was necessary as the MOFScript does not provide a function
	 * on its own to produce random Integer values. The values are
	 * used to pick random values from pools/collections.
	 */
	public static void main(String[] args) {
	}
	public RandomIntGenerator(){}
	/*
	 * Method getRandomInt produces a random Integer value,
	 * given a seed value. The produced random value is returned
	 * to the caller. This function is called from the MOFScript
	 * files in the current version of the mbt-tool.
	 * @param upperLimit int, the seed value
	 * @param generator Random, the generator object
	 * @return ret int, the derived random Integer
	 */
	public int getRandomInt(int upperLimit){
		int ret=0;
		Random generator = new Random();
		ret = generator.nextInt(upperLimit);
		return ret;
	}

}
```

9.13 Appendix M– Sample test-case

```
<%
from jpype import *
import os.path
import jpype
sys.path.append(os.path.basename(os.path.abspath(__file__)))
sys.path.append('.')
try:
	import Get_Val
except:
	print os.path.basename(os.path.abspath(__file__))
	import Get_Val
attTable = []
attTable.extend(['SystemUnit','systemunit','systemUnit','NumberOfActiveCalls','Integer','1','s
tatus.SystemUnit.State.NumberOfActiveCalls',[]])
attTable.extend(['SystemUnit','systemunit','systemUnit','MaximumNumberOfCalls','Integer','1','
status.SystemUnit.State.maximumNumberOfCalls',[]])
attTable.extend(['Conference','conference','conference','PresentationMode','String','1','statu
s.Conference.Presentation.Mode',[]])
attTable.extend(['Call','call0','calls','CallRate','Integer','3','CallRate',['status.Call']])
attTable.extend(['Call','call1','calls','CallRate','Integer','3','CallRate',['status.Call']])
attTable.extend(['Call','call2','calls','CallRate','Integer','3','CallRate',['status.Call']])
attTable.extend(['IncomingVideoChannel','incomingvideochannel0','incomingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','IncomingVideoChannel']]
)
attTable.extend(['IncomingVideoChannel','incomingvideochannel1','incomingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','IncomingVideoChannel']]
)
attTable.extend(['IncomingVideoChannel','incomingvideochannel2','incomingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','IncomingVideoChannel']]
)
attTable.extend(['IncomingVideoChannel','incomingvideochannel3','incomingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','IncomingVideoChannel']]
)
attTable.extend(['IncomingVideoChannel','incomingvideochannel4','incomingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','IncomingVideoChannel']]
)
attTable.extend(['IncomingVideoChannel','incomingvideochannel5','incomingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','IncomingVideoChannel']]
)
attTable.extend(['OutgoingVideoChannel','outgoingvideochannel0','outgoingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','OutgoingVideoChannel']]
)
attTable.extend(['OutgoingVideoChannel','outgoingvideochannel1','outgoingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','OutgoingVideoChannel']]
)
attTable.extend(['OutgoingVideoChannel','outgoingvideochannel2','outgoingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','OutgoingVideoChannel']]
)
attTable.extend(['OutgoingVideoChannel','outgoingvideochannel3','outgoingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','OutgoingVideoChannel']]
)
attTable.extend(['OutgoingVideoChannel','outgoingvideochannel4','outgoingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','OutgoingVideoChannel']]
)
attTable.extend(['OutgoingVideoChannel','outgoingvideochannel5','outgoingVideoChannel','Protoc
ol','VideoProtocol','6','Video.Protocol',['status.MediaChannels.Call','OutgoingVideoChannel']]
)

# Initialize helper lists
attClass = attTable[0::8]
attNames = attTable[3::8]
attObject = attTable[1::8]
attPath = attTable[6::8]
attRole = attTable[2::8]
attMult = attTable[5::8]
attSubpath = attTable[7::8]
attType = attTable[4::8]

classpath = os.path.join(os.path.abspath('.'))
jpype.startJVM("jvm.dll", "-Djava.ext.dirs=%s" % classpath)
Test = jpype.JClass('Test.TestData')
```

```
ClassDiagram = jpype.JClass('Test.ClassDiagramTestData')
t = Test()
c = ClassDiagram()
results = []
init_var('tng_t',t)
init_var('tng_c',c)
init_var('tng_attTable',attTable)
# Initial State: Start
# Transition: Start->Idle     Guard:
# State: Idle
# StateInvariant:IdleStateInvariant  Value: self.systemUnit.NumberOfActiveCalls = 0 and
self.conference.PresentationMode = 'Off'
eval = 'IdleStateInvariant'
# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
        expression = attRole[index]+'.'+name
        if int(attMult[index]) > 1:
                value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
        else:
                value = Get_Val.new_get_value(target,attPath[index])
        if len(value)-1 == 0:
                j = 0
        if expression in "self.systemUnit.NumberOfActiveCalls = 0 and
self.conference.PresentationMode = 'Off'":
                existAtt += " " + attClass[index]
                existAtt += " " + name
                existAtt += " " + attObject[index]
                existAtt += " " + attType[index]
                theValue = " '"  + value[j] + "'"
                if attType[index] == 'Integer':
                        theValue = " " + value[j]
                existAtt += theValue
        if j < (len(value)-1):
                j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls = 0 and
self.conference.PresentationMode = 'Off'")
print "Result of query: ", queryResult
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
# Transition: Idle->Connected Guard: self.systemUnit.NumberOfActiveCalls = 0
# Start Trigger: dialSaturn, Type: MethodCall
eval = 'dialSaturn'

# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
        expression = attRole[index]+'.'+name
        if int(attMult[index]) > 1:
                value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
        else:
                value = Get_Val.new_get_value(target,attPath[index])
        if len(value)-1 == 0:
                j = 0
        if expression in "self.systemUnit.NumberOfActiveCalls = 0":
                existAtt += " " + attClass[index]
                existAtt += " " + name
                existAtt += " " + attObject[index]
                existAtt += " " + attType[index]
                theValue = " '"  + value[j] + "'"
                if attType[index] == 'Integer':
                        theValue = " " + value[j]
                existAtt += theValue
        if j < (len(value)-1):
                j = j+1
```

```
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls = 0")
print "Result of query: ", queryResult
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
init_var('tng_results',results)
init_var ('caller', t.getNumberToConnect("testtarget"))
%>
       testtarget.dial $caller SIP  2b
       wait 5

<%
try:
       import Get_Val
except:
       print os.path.basename(os.path.abspath(__file__))
       import Get_Val
attTable = get_var('tng_attTable')
t = get_var('tng_t')
c = get_var('tng_c')
results = get_var('tng_results')

# Initialize helper lists
attClass = attTable[0::8]
attNames = attTable[3::8]
attObject = attTable[1::8]
attPath = attTable[6::8]
attRole = attTable[2::8]
attMult = attTable[5::8]
attSubpath = attTable[7::8]
attType = attTable[4::8]

init_var('tng_attTable',attTable)
init_var('tng_t',t)
init_var('tng_c',c)
# Finish Trigger: dialSaturn, Type: MethodCall
# State: Connected
# StateInvariant:Connected    Value: self.systemUnit.NumberOfActiveCalls=1 and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off or
c.outgoingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off)->size() =0
eval = 'Connected'
# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
       expression = attRole[index]+'.'+name
       if int(attMult[index]) > 1:
              value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
       else:
              value = Get_Val.new_get_value(target,attPath[index])
       if len(value)-1 == 0:
              j = 0
       if expression in "self.systemUnit.NumberOfActiveCalls=1 and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size() =0  ":
              existAtt += " " + attClass[index]
              existAtt += " " + name
              existAtt += " " + attObject[index]
              existAtt += " " + attType[index]
              theValue = " '"  + value[j] + "'"
              if attType[index] == 'Integer':
                     theValue = " " + value[j]
              existAtt += theValue
       if j < (len(value)-1):
              j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls=1 and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size() =0")
```

```
print "Result of query: ", queryResult
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
# Transition: Connected->NotFull     Guard: self.systemUnit.NumberOfActiveCalls = 1
# Start Trigger: EndPointConnect, Type: SignalReception
eval = 'EndPointConnect'

# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
        expression = attRole[index]+'.'+name
        if int(attMult[index]) > 1:
                value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
        else:
                value = Get_Val.new_get_value(target,attPath[index])
        if len(value)-1 == 0:
                j = 0
        if expression in "self.systemUnit.NumberOfActiveCalls = 1":
                existAtt += " " + attClass[index]
                existAtt += " " + name
                existAtt += " " + attObject[index]
                existAtt += " " + attType[index]
                theValue = " '"  + value[j] + "'"
                if attType[index] == 'Integer':
                        theValue = " " + value[j]
                existAtt += theValue
        if j < (len(value)-1):
                j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls = 1")
print "Result of query: ", queryResult
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
init_var('tng_results',results)
init_var ('caller',t.getCaller("dial"))
%>
        $caller.dial testtarget H323  1b
        wait 5

<%
try:
        import Get_Val
except:
        print os.path.basename(os.path.abspath(__file__))
        import Get_Val
attTable = get_var('tng_attTable')
t = get_var('tng_t')
c = get_var('tng_c')
results = get_var('tng_results')

# Initialize helper lists
attClass = attTable[0::8]
attNames = attTable[3::8]
attObject = attTable[1::8]
attPath = attTable[6::8]
attRole = attTable[2::8]
attMult = attTable[5::8]
attSubpath = attTable[7::8]
attType = attTable[4::8]

init_var('tng_attTable',attTable)
init_var('tng_t',t)
init_var('tng_c',c)
# Finish Trigger: EndPointConnect, Type: SignalReception
# State: NotFull
# StateInvariant:Not_Full     Value: (self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls<self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off or
c.outgoingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off)->size()=0
eval = 'Not_Full'
```

```
# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
	expression = attRole[index]+'.'+name
	if int(attMult[index]) > 1:
		value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
	else:
		value = Get_Val.new_get_value(target,attPath[index])
	if len(value)-1 == 0:
		j = 0
	if expression in "(self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls<self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size()=0":
		existAtt += " " + attClass[index]
		existAtt += " " + name
		existAtt += " " + attObject[index]
		existAtt += " " + attType[index]
		theValue = " '"  + value[j] + "'"
		if attType[index] == 'Integer':
			theValue = " " + value[j]
		existAtt += theValue
	if j < (len(value)-1):
		j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"(self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls<self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size()=0")
print "Result of query: ", queryResult
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
# Transition: NotFull->NotFull_1     Guard: self.systemUnit.NumberOfActiveCalls >2 and
self.systemUnit.NumberOfActiveCalls < self.systemUnit.MaximumNumberOfCalls -1
# Start Trigger: callDisconnectSaturn, Type: MethodCall
eval = 'callDisconnectSaturn'

# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
	expression = attRole[index]+'.'+name
	if int(attMult[index]) > 1:
		value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
	else:
		value = Get_Val.new_get_value(target,attPath[index])
	if len(value)-1 == 0:
		j = 0
	if expression in "self.systemUnit.NumberOfActiveCalls >2 and
self.systemUnit.NumberOfActiveCalls < self.systemUnit.MaximumNumberOfCalls -1":
		existAtt += " " + attClass[index]
		existAtt += " " + name
		existAtt += " " + attObject[index]
		existAtt += " " + attType[index]
		theValue = " '"  + value[j] + "'"
		if attType[index] == 'Integer':
			theValue = " " + value[j]
		existAtt += theValue
	if j < (len(value)-1):
		j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"self.systemUnit.NumberOfActiveCalls >2 and
self.systemUnit.NumberOfActiveCalls < self.systemUnit.MaximumNumberOfCalls -1")
print "Result of query: ", queryResult
results.extend([eval,queryResult])
```

```
# Finisehed Temporary Consistency
init_var('tng_results',results)
init_var ('caller', t.getNumberToDisconnect("testtarget"))
%>
        testtarget.disconnect $caller
        wait 5

<%
try:
        import Get_Val
except:
        print os.path.basename(os.path.abspath(__file__))
        import Get_Val
attTable = get_var('tng_attTable')
t = get_var('tng_t')
c = get_var('tng_c')
results = get_var('tng_results')

# Initialize helper lists
attClass = attTable[0::8]
attNames = attTable[3::8]
attObject = attTable[1::8]
attPath = attTable[6::8]
attRole = attTable[2::8]
attMult = attTable[5::8]
attSubpath = attTable[7::8]
attType = attTable[4::8]

init_var('tng_attTable',attTable)
init_var('tng_t',t)
init_var('tng_c',c)
# Finish Trigger: callDisconnectSaturn, Type: MethodCall
# State: NotFull
# StateInvariant:Not_Full     Value: (self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls<self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off or
c.outgoingVideoChannel->asSequence()->last().Protocol <> VideoProtocol::Off)->size()=0
eval = 'Not_Full'
# Start Temporary Consistency
existAtt = ""
value = []
j = 0
# force reload of status document
target = get_system("testtarget")
target.status.forcetext
for index,name in enumerate(attNames):
        expression = attRole[index]+'.'+name
        if int(attMult[index]) > 1:
                value = Get_Val.new_get_value(target,attPath[index],attSubpath[index])
        else:
                value = Get_Val.new_get_value(target,attPath[index])
        if len(value)-1 == 0:
                j = 0
        if expression in "(self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls<self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size()=0":
                existAtt += " " + attClass[index]
                existAtt += " " + name
                existAtt += " " + attObject[index]
                existAtt += " " + attType[index]
                theValue = " '"  + value[j] + "'"
                if attType[index] == 'Integer':
                        theValue = " " + value[j]
                existAtt += theValue
        if j < (len(value)-1):
                j = j+1
print "ExistAtt: ",existAtt
queryResult = c.updateObjectDiagram(existAtt,"(self.systemUnit.NumberOfActiveCalls>1 and
self.systemUnit.NumberOfActiveCalls<self.systemUnit.MaximumNumberOfCalls) and
self.conference.PresentationMode = 'Off' and self.conference.calls->select(c:Call |
c.incomingVideoChannel->asSequence()->last().Protocol <> 'Off' or c.outgoingVideoChannel-
>asSequence()->last().Protocol <> 'Off')->size()=0")
print "Result of query: ", queryResult
```

```
results.extend([eval,queryResult])
# Finisehed Temporary Consistency
print("Evaluation of OCL-expressions:")
for res in results:
       print(res)
%>
       testtarget.disconnect
```