Using TTCN-3 for testing Platform Independent Models

Gabor Batori, Domonkos Asztalos
Software Engineering Group, Ericsson Hungary Ltd.
P.O.B.107, H-1300 Budapest, Hungary
Phone: +36-1-437 7537, Fax: +36-1-437 7767
{Gabor.Batori,Domonkos.Asztalos}@ericsson.com

Abstract. In the field of telecommunication UML and Model Driven Architecture (MDA) have an increasing acceptance. MDA brings up new questions about the testing of the application developed by this technology. In MDA, Platform Independent Model (PIM) is the source of the system, and all maintenance and enhancement is performed at the platform independent level. However, MDA supporting tools provide only limited means for describing model level test procedures so a framework for model testing is indispensable. This paper investigates how to assist the model level test development with TTCN-3. We found that with the help of model translators we can facilitate and partly automate the test development process.

1 Introduction

The Model Driven Architecture (MDA) [1] of the Object Management Group has become the dominating trend in software engineering. MDA recommends starting the design of an application with a Platform-Independent Model (PIM) representing the business functionality and behavior, undistorted by technology details in the form of a UML model. In the next phase, Platform-Specific Models (PSM) containing software architecture dependent information are generated from the PIM by applying mappings in an MDA tool, preferably by automatic model transformations. Finally, in the code generation phase MDA tools automatically generate all or most of the implementation code for the deployment technology. Model transformation methodologies have been under extensive research recently. These transformation techniques provide higher quality compared to manually written programs but they require that the PIM contains the smallest possible number of faults. Unfortunately, it does not matter what technology we use and how much time we put into design and how careful we are when programming; mistakes are inevitable. Automation does not alone guarantee neither the proper choice of underlying architecture nor the elimination of conceptual flaws from the analysis model because defects injected in the requirements analysis are also deployed automatically into the implementation.

Due to the increased complexity of IT systems and increased customer requirements for quality of service (QoS) and reliability, mathematical-based test generation techniques often fail, because of the difficulty to select test cases from a (theoretical) unbounded number of tests. However, there is a strong need for effective testing of complex applications, because it is a well known fact that the development and implementation of tests is very time consuming and labor intensive. MDA based software development offers an effective way to analyze computer systems with early-phase simulation and the tests created at the early-phase analysis can be reused on the implementation level.

We use TTCN-3 [2] as a test description language for platform independent model tests. One essential benefit of TTCN-3 is that the specification of tests is possible in a platform independent way. Our goal is to develop a framework for testing Platform Independent Model with TTCN-3, and analyze the possibility of reusing the analysis level tests on the implementation level.

This paper is organized as follows. In Section 2, we examine existing researches related to testing UML models. We will present a brief review of Model Driven Architecture focusing on the testing concepts in Section 3 and address our testing approach. Section 4 concludes the TTCN-3 language architecture and its relation to MDA. In Section 5 we present the structure of a model testing framework and the generation of this framework with a model transformer. In Section 6, we summarize the current status of the tester and our future plans.

2 Related work

Lots of approaches have been taken to use the early-phase model as a basis of test development. Classic problems of model-based testing are [3]:

1. the generation of test cases from model according to a given coverage criterion,
2. the generation of a test oracle to determine the expected result of a test,
3. the execution of tests in test environments, possibly also generated from models.

Model-based testing is used to define tests which verify that a specific implementation accurately capture its requirements. Algorithms [4,5,6] have been defined to derive tests from formal system specification given in UML notation and their usage has been demonstrated with sample applications. But today none of the approaches are widely used in the industrial practice for large applications. One reason may be the difficulty to define selection criterions that result test cases with high coverage in respect to the requirements of the application. Furthermore, if MDA and code generation techniques are used, the test generation can apply with a purpose different of the classical approach. The difficulty that restricts the usage of problem (1) is: the code and test generation algorithms have the same source, the *PIM*. In this case only the correctness of the translation method could be verified.

UML technology focuses primarily on the definition of system structure and behavior and provides only limited means for describing complex test procedures [7]. CASE tools provide only minimal support for developing tests. They only assist to create unit tests, therefore:

– We can execute only a small number of tests.
– We have to execute and estimate them manually.
– The scope of a test is only an object or a small cluster of objects.

A special UML profile based on the UML 2.0 specification was initiated for test description using UML [8]. This profile aims at bridging the gap between designers and testers by providing a means for using UML for system modeling and test development. This allows a re-use of UML design documents for testing and makes test development possible in an early system development phase. But UML is not the appropriate language to address executable tests, because it is hard to define complex structures of test data and the graphical notation is sometimes inconvenient especially in case of a complex test description. The authors of paper [9] showed a methodology of how to use the UML 2.0 Testing Profile on an existing UML design model. The usability of the method was demonstrated by developing a test model for a Bluetooth roaming model.

The paper [10] describes a MOF (Meta-Object Facility) based meta-model of TTCN-3 and the realization of the meta-model in Eclipse. Moreover, it shows how to integrate TTCN-3 tools via this meta-model.

3 Testing concepts in MDA

MDA envisages systems being designed independently of the eventual technologies, and a PIM can then be transformed into specific platforms. This section provides an overview of the model driven architecture focusing on the testing aspect.

3.1 Software development with Executable UML

The OMG Model Driven Architecture addresses the complete life cycle of designing, deploying, integrating and managing applications using open standards. The MDA aims at providing a framework for the creation of applications in such a context where even the interface between the target application and the underlying execution platform is changing. MDA is a new way of writing specifications and developing applications, based on a platform-independent model (PIM) and using transformations to create platform-specific models (PSMs) and source code. The idea is that in a platform independent model the developer concentrates on a description of *what* the system has to do without going into details of *how* that will be achieved. The platform specific model, by contrast, describes how the system will realize the behavior implied by the analysis model [11]. MDA uses the Unified Modeling Language (UML) as notation. The UML 1.4 standard had relatively little to say about the detailed behavior that might be specified for the action associated with transitions and states or the methods implementing operation. In UML 1.5 and UML 2.0 specification, a UML Action Semantics [12] has been introduced. With the Action Semantics (AS) we can create executable models [13] with a detailed dynamic behavior description. This model can be executed in an appropriate simulator. The benefits of this approach go well beyond simply reducing or eliminating the coding stage. It also ensures platform independence, avoids obsolescence (programming languages may change, the model does not) and allows full verification of the models by executing them in a test and debug environment.

3.2 Testing in MDA

The designated architecture of MDA is summarized in Fig. 1. Firstly, analysts create the analysis model[1] based on the system requirements. Then automatic transformations are

[1] Analysis model and Platform Independent Model (PIM) are used as synonyms in this paper.

used to create platform specific models (PSMs) and source code. The last phase is the testing of the implementation. In this method the testing phase only starts after the code generation has finished. There are two main problems in this method: (1) The creation of a new model transformer for a new platform is very time consuming, hence the code generation and the testing phase can be delayed, although there is an executable and testable model. (2) The model transformation can lead to the mixing up of platform independent and platform specific information in the implementation. This makes it difficult to eliminate the errors from the PIM.

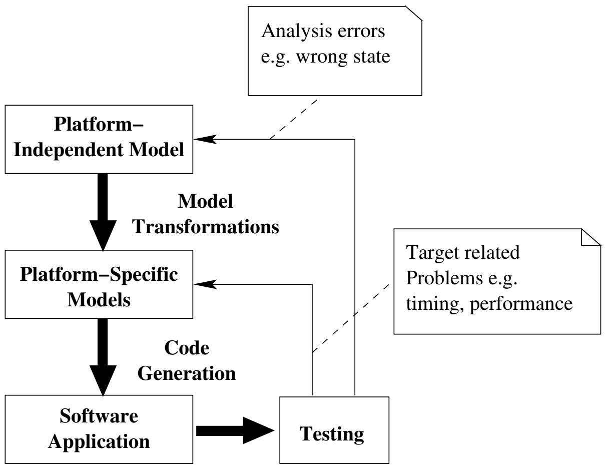

Fig. 1. The MDA architecture

In our approach (Fig. 2), we split the testing into two phases. In the first phase, the simulated platform independent model is verified. In this early stage only the functional correctness of the model could be tested. Since the analysis model is the source of the system and the following model transformations, it requires rigorous testing. The errors found during this phase are related to the analysis model, therefore we call them analysis errors.

In the second phase, the testing of the implementation is started. Based on the early-phase tests the testers can build performance, inter-operability etc. test cases. The functional tests can be also repeated in order to verify that the model transformations do not make any unexpected changes. To minimize the work invested to the testing of the application we should reuse the early-phase tests. In order for testing to reach its full potential, it is essential to use the same testing framework throughout the entire MDA software development process. We use a dashed arrow between the implementation testing and the platform independent model in Fig. 2 because the early-phase testing ensures that the implementation does not contain analysis errors, hence during the implementation testing only platform related errors can be found. We present some exceptions in the end of Section 5.2

In the following section we demonstrate that the TTCN-3 language is a feasible candidate for this purpose. We present a short overview of the standardized language for test description, focusing on how to depict tests on analysis as well as on implementation level.

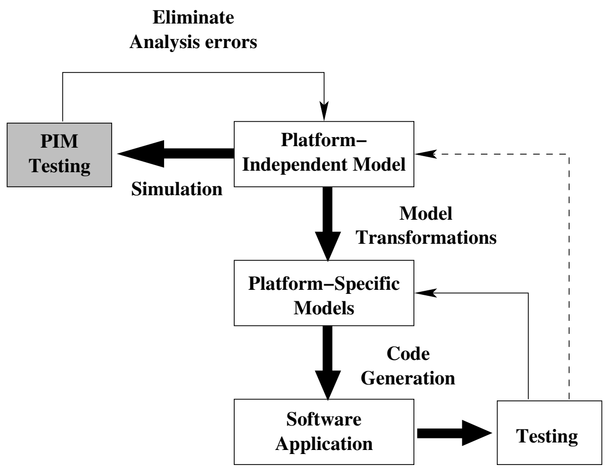

Fig. 2. Extended test model in MDA

4 TTCN-3 and its relation to MDA

TTCN-3 (Testing and Test Control Notation 3) is the new industry-standard test specification language that was developed and standardized by the European Telecommunication Standards Institute (ETSI). TTCN-3 can be applied for all kinds of black-box testing for reactive and distributed systems and makes it possible to be used not only in conformance testing of telecommunication protocols but as well as for testing Internet, mobile, data base access etc. protocols and also for inter-operability, robustness etc. testing. Use of TTCN-3 to support test development has been investigated to encourage the parallel development of a test suite together with a standard system analysis. TTCN-3 language consists of three main units:

Test Behavior Test behavior is a specification of what to test with which input, result, and under which conditions. The TTCN-3 language defines several constructs for describing the functionality of a test system. TTCN-3 allows an easy and efficient description of complex test behavior in terms of sequences, alternatives, loops and parallel stimuli and responses.

Test Configuration This part is responsible for the communication between the System Under Test (SUT) and the test system. However, the real physical connection is outside the scope of TTCN-3. Instead, a well defined (but abstract) test system interface shall be associated with each test case. A complex test configuration may contain several test components which could communicate with each other and the system under test.

Test Data Definition One of the key elements of TTCN-3 is the ability to send and receive complex messages over the communication ports defined by the test configuration. TTCN-3 supports a number of predefined basic data types and structured types constructed from the basic data types. The TTCN-3 has a special language element, the template, that provides sophisticated means for describing test data. Templates are used either to transmit a set of distinct values or to test whether a set of received values match the template specification.

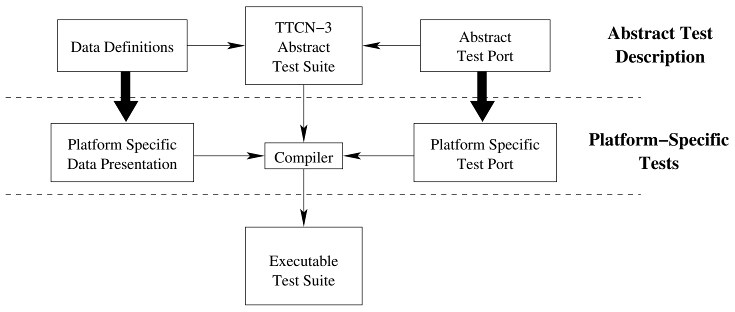
Data Definitions
TTCN−3 Abstract Test Suite
Abstract Test Port
Abstract Test Description
Platform Specific Data Presentation
Compiler
Platform Specific Test Port
Platform−Specific Tests
Executable Test Suite

Fig. 3. The Architecture of the TTCN-3 language

The general testing process with TTCN-3 includes the following main steps: the developed abstract test suite is compiled and extended with an adaptor (one special implementation of an abstract TTCN-3 test port) that provides the connection between the tested system and the executable test suite. Then, the executable test suite is executed against the system under test. Finally, the results are evaluated. The TTCN-3 is an abstract language, hence one can describe the test behavior independently of the underlying communication architecture and data presentation. The structure of a TTCN-3 tester is summarized in Fig. 3. Note that the basic conception of the model driven architecture is almost the same (see in Section 3.2), but the TTCN-3 focusing on the testing domain.

There are two ways to alter the behavior of a test suite. One solution is to change the communication interface and the data encoding/decoding rules. In the field of wireless communication there are many protocols that are able to transmit data in several different ways depending on how reliable the connection is or how important the message is etc. A good example is the WAP (Wireless Application Protocol) protocol, which can work on various bearers i.e. SMS, GPRS, Circuit Switched Data etc. This functionality is especially important in the 3G or 4G mobile technologies where many high level applications have to work on different transaction protocols.

The second solution is to change the test data definitions. TTCN-3 provides a simple form of inheritance that enables us to modify an existing template without changing the original definition. This makes the adaptation of templates to different testing situations possible and avoids the duplication of similar test data.

In accordance with the discussion above this approach allows to use TTCN-3 during all part of the model-driven software development. The functionality and the specification details are separated, therefore the early-phase functional tests can be reused on implementation level.

5 Testing framework for Platform Independent Models with TTCN-3

In this section, we show how to use the TTCN-3 language in model driven software development.

5.1 Simulating executable UML models

Model execution enables developers to focus on the appropriate behavior of the problem to be solved, independently of platform dependent problems at an early development phase. Executable models allow the early verification with simulation, since they completely describe the dynamic behavior of the system. In order to simulate a model we need a special environment that is capable to interpret executable models. This environment is referred to as a *UML Virtual Machine*. As input, the Virtual Machine requires an executable UML model (class diagram, state-chart, action specification) and executes the model according to the initial state and the receiving inputs. Having this Virtual Machine we are able to define test interfaces.

According to [17], PIMs suffer from testability problems in the area of *observability*, the ability to detect errors in control flows, and *controllability*, the capability to cause the software to execute an appropriate path. An OMG Request For Proposal (RFP) has been initiated on a standardized interface of testing and debugging executable UML models [16]. In Fig. 4, the structure of the interface is concluded.

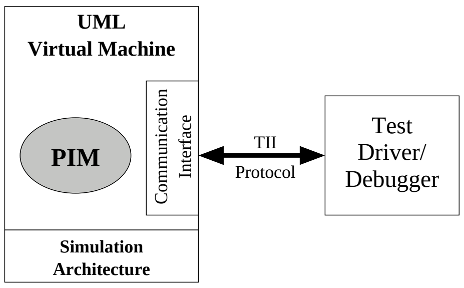

Fig. 4. Test Instrumentation Interface

The goal of the Test Instrument Interface is to standardize the hooks into model execution to allow test setup, stimulus, and data collection. Model simulators have some support for model connection, they provide the ability to define breakpoints in the executable model, to log the actions during the execution. But they do not allow to use external testers for the testing of the model. In our approach, we transform UML models to be able to communicate with an external tester, so that the original simulation framework can be used. The interface definitions between the test system and the UML model are addressed on UML level and the physical communication interfaces are derived from these definitions. The extension of executable models with communication ability is provided through model transformations.

5.2 Towards the mapping of UML models into TTCN-3

To create executable tests three partly coherent tasks have to be carried out:

– Define the data uses in the test cases
– Define the behavior of the test cases
– Define the test framework (test configuration)

In this paper we concentrate on the solution of the first and the third problem with the help of a model translators. A test framework covers the concepts for specifying test components, the interfaces of and connections between the test components to the System Under Test. Telecommunication protocols and softwares are distributed applications, therefore our testing framework was designed to allow the definition of complex distributed test scenarios on model level. We defined a model transformer which is capable to extend an executable UML model in order to test in a distributed test configuration.

Most tedious activities during test development are to accurately define the interfaces between the system under test (SUT) and the test system, and specify the test data sending and receiving on these interfaces. Therefore, another model transformer was defined to create the data definition and the testing framework in TTCN-3 core language. The test cases can be written manually based on the derived definition.

MDA offers the potential to automatically transform a PIM, perhaps after annotating it with some platform information, to different PSMs. Modeler will *tag* their PIM component with information to control the translation. This approach allows us to store test specific information in the analysis model independently of the design aspects.

To create a model transformer in UML we have to create the meta-model of the target language, in our case the meta-model of the TTCN-3 language. Fig. 5 depicts the communication and data representation part of our meta-model.

The elements of the meta-model are populated (instantiate the elements of the meta-model) depending on the platform-independent model. According to our experience the communication interfaces on the implementation level are represented by operations of classes on the model level. Hence, we specify a *tagging* structure in order to mark the operations that are relevant for testing. *Tags* may denote the direction of the communication channel created from the *tagged* operation or the name of the test port which the given operation belongs to. The data presentation of UML differs from the one of TTCN-3, therefore we had to define mapping rules between them. Because of the lack of space only the main mapping rules are summarized in Table 1.

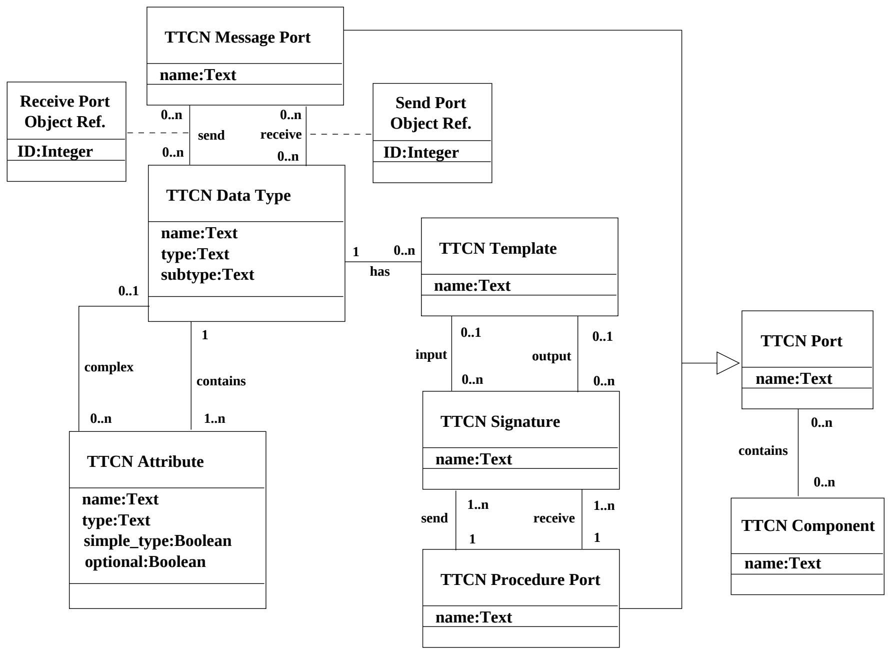

Fig. 5. Meta-model of the TTCN-3

Table 1. Data presentation mapping rules

UML	TTCN
Simple types (Boolean, Double,Integer)	Simple TTCN-3 types
Text	charstring
Data set	record of
Operation parameters	record
Polymorphic operations (with small changes)	optional parameters
Polymorphic operations	union types

The simple UML data types have unambiguous representation in TTCN-3. The only exception is the text type because in TTCN-3 five different basic string types can be defined. We selected the charstring type to represent the UML text type in TTCN-3.

We map the input and output parameters of the operations into record types. TTCN-3 ports are also generated which allows to send and receive these record types. We can define sending templates for these records to test the operation with various input parameters. In addition, we can define receiving templates to automatically verify the results of the operation using the TTCN-3 matching mechanism.

The last two rows of Table 1 show an example how the structure of the platform independent model influence the TTCN-3 data presentation. If analysts create generalization relations with many sub-classes and with polymorphic operations then the structure of the derived TTCN-3 data types have to reflect this inheritance tree. An operation of the parent-class can be the representation of a communication port and the sub-classes inherit this operation but in some sub-classes the operation is overridden. In this case some parameters of the operation may become *optional* parameters in TTCN-3 if only small changes (one or two parameters appear or disappear in the operation definition) occurred during the redefinition of the operation. If the changes in the parameters of the operation are considerable then it is more profitable to create a new *record* for this parameter structure. In order to refer that the new *record* is derived from a parent-class we compose a *union* type which contains the different definitions of the records corresponding to the operation.

To achieve testability, we also use the *tagged* elements of the model as weaving points where we should insert new instructions to extend the UML model. The extended model is capable of communicating with a tester in the simulator. The extension is based on the definition of the tagged operation.

With MDA we can develop a translator model [14] which is capable to collect information from high level, platform independent models and generate the TTCN-3 test interfaces and data definitions. Accordingly, the technical problems related to the communication between the test system and the UML Virtual Machine can be hidden from the testers as well as the analysts. Fig. 6 depicts the structure of the PIM tester. A distributed client-server based environment is responsible for the communication between the two parts of the model tester. This communication interface is also generated from the analysis model. The interface has two part. The first part is running in the UML Virtual Machine. This part is capable to access the model. The second part is the implementation of TTCN-3 test ports. This implementation contains the mappings between the UML and TTCN-3 data types.

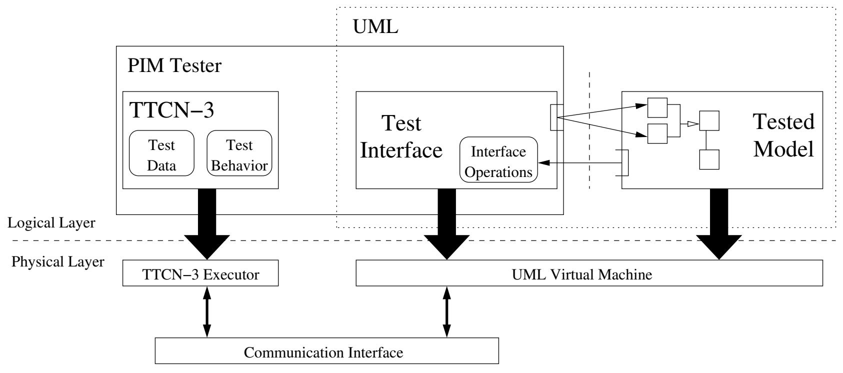

Fig. 6. Structure of the PIM tester

The different parts of the platform independent model testing are summarized in Fig. 7. The test development is started with the transformation of the PIM. The TTCN-3 translator creates the communication interfaces and the data definitions. Based on this definitions the test data and the test behavior can be defined. The model translator creates the extended PIM, which is executed in a simulator. The TTCN-3 test cases are executed on the simulated extended platform independent model.

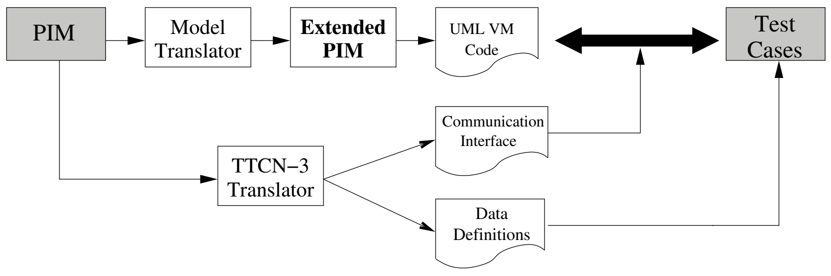

Fig. 7. Mapping to TTCN-3

The model-based testing usually not enough to eliminate all faults from the software because of the following reasons:

– The model may contain special object structures.
– Usage of native codes in the model.
– Usage of third party libraries, existing components.

There are special object structures [15] whose functions depend on the architecture, hence the functions of these objects have to be tested on implementation level as well. Some MDA tools allow to insert INLINE (platform specific, native language) codes into the body of the platform independent action code that can be tested only after the mapping to the platform specific implementation occurs. Furthermore, one can use third party libraries or existing components that were created without model driven technology. In this case, the integration with these components have to be tested, but it is only possible on implementation level. In spite of these limitations of the platform independent testing, according to our experience approximately 50-60% of the errors can be found and eliminated in analysis phase.

5.3 Testing through a MDA software development

We experimented on our testing framework during the development of a network management software. The test environment is depicted in Fig. 8. The test architecture consists of three different elements.

Managed Network A managed network may contain a few or several hundreds of managed nodes (MN). The managed nodes provide support to ATM switching and IP forwarding system. An arbitrary mix of different traffic types – data, voice, and video type of traffic – can be handled with preserved quality of service and with efficient use of bandwidth for each traffic type.

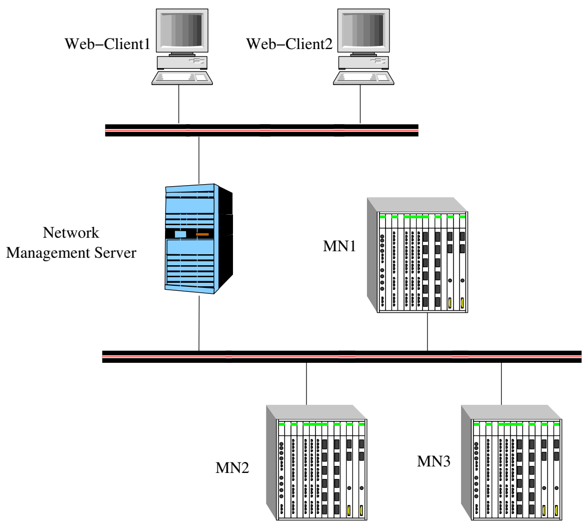

Fig. 8. Test architecture

Network Management Server This application is required by telecommunication operators for providing reliable operations of the communication network. The tasks involved include monitoring, troubleshooting, and control operations in a wide range of network management areas.

Web clients The operators of the network are able to access the management software through web-based clients. In case of error the operators can reconfigure the network topology manually.

The main component, the management server, is modeled in UML and the other components (client, managed nodes) are emulated by TTCN-3 components. This heterogeneous infrastructure can be tested with TTCN-3 parallel components. Our goal hereby is to test the functionality of the server with different network structures or with erroneous network topologies. In case of a complex real network it is difficult to configure the network to generate incorrect answers. With TTCN-3 and simulation we can easily establish these situations and can verify that our application (the simulated PIM model) works as we expect.

A typical problem in model driven development is that the development of the platform-independent model finishes before the development of the transformation rules for the specific platform would be completed. In this case we can test the PIM in a simulator but the test environment act as a real network.

5.4 Empirical Experiences

We used a sample TTCN-3 test module to investigate what kind of modifications were needed to rerun the early-phase tests on the implementation. At first, we defined manually 20 test cases to verify the main functionality of the network management application. The test module contained 25 type definitions and 30 template specifications for the data types. Two types of TTCN-3 test port were used during the testing: a HTTP-based test port for the client and a SNMP port for the communication to the managed nodes. For simulation testing purposes the test ports, the test components and the data type definitions were automatically generated from the PIM model. The test port implementations for the Ericsson's TTCN-3 test executor were also generated from the PIM model. Three parallel test components were used during the testing, one for the emulation of the web-client and two for the emulation of the managed network. The test suite was executed against the simulated model and 5 errors were found in the PIM.

Secondly, we executed this test suite on the implementation. The implementation was generated from the platform independent model with a model transformer developed in Ericsson Hungary. Naturally, we had to change the implementation of the test ports. We also needed one new data type and 2 new templates. With these modifications every test case could be executed on the implementation. One additional error was found in the implementation which was caused by an integration problem between an existing and a newly developed component.

6 Conclusion and future work

In this article we propose an approach for model level testing of applications designed with model-driven technology. We can adapt this test design process into the standard model-driven software development process. By using this approach, we are able to analyze Platform Independent Models with tests written in a standardized test description language. These early-phase tests primarily focus on the functional correctness of the software. Moreover, by extending the platform independent tests, other types of tests (e.g. inter-operability, performance) can be derived. Accordingly, the implementation level test development time can be reduced.

Regarding further investigation, it would be interesting to study the possibility of using this testing concept throughout the entire model-driven software development process and work out a general *Model-Driven Test Development* method.

References

1. R. Soley: Model Driven Architecture: An Introduction. http://www.omg.org/mda.
2. ETSI ES 201 873-1: The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. V2.2.1 (2003-02), 2003; also an ITU-T standard Z.140.
3. R.Heckel, M.Lohmann: Towards Model-Driven Testing, Electronic Notes in Theoretical Computer Science Vol.82 (6), 2003.
4. J. Hartman, C. Imoberdorf, M. Meisinger: UML-Based Integration Testing, ISSTA 2000.
5. J. Offut, A. Abdurazik: Generating Tests from UML Specification, UML99 Fort Collins (CO), October 1999.
6. L. C. Briand , Y. Labiche: A UML-Based Approach to System Testing, Journal of Software and Systems Modeling (SoSyM) Vol. 1 No.1 2002 pp. 10-42.
7. I. Schieferdecker, Z. R. Dai, J. Grabowski, A. Rennoch: The UML 2.0 Testing Profile and its relation to TTCN-3, Testing of Communicating Systems – 15th IFIP International Conference, TestCom2003, Sophia Antipolis (F), May 2003. Lecture Notes in Computer Science (LNCS) 2644, Springer, May 2003.

8. UML Testing Profile (Final Submission), April 2004 http://www.fokus.gmd.de/u2tp/.
9. Z. R. Dai, J. Grabowski, H. Neukirchen, H. Pals: From Design to Test with UML – Applied to a Roaming Algorithm for Bluetooth Devices. Testing of Communicating Systems – 16th IFIP International Conference, TestCom2004, Oxford, United Kingdom, March 2004. Lecture Notes in Computer Science (LNCS) 2978, Springer, March 2004.
10. I. Schieferdecker, G. Din: A Meta-model for TTCN-3, Applying Formal Methods: Testing, Performance, and M/E-Commerce: FORTE 2004 Workshops, Toledo, Spain. Lecture Notes in Computer Science (LNCS) 3236, Springer, October 2004.
11. S. Shlaer, S. J. Mellor: Recursive Design of an Application-Independent Architecture, IEEE Software, pp. 61-72, January/February 1997.
12. I. Wilkie, A. King, M. Clarke, C Raistrick: UML ASL Reference Guide, Kennedy Carter, 2001.
13. Supporting Model Driven Architecture with eXecutable UML Kennedy Carter 2002.
14. I. Wilkie, A. King, M. Clarke, C Raistrick: The Intelligent OOA Strategy for Configurable Code Generation, Kennedy Carter, 1997.
15. S. Shlaer, N. Lang: Shlaer-Mellor Method: The OOA96 Report. http://www.projtech.com.
16. Model-level Testing/Debug RFP (Final Submission) April 2004 http://www.omg.org.
17. G. Eakman: Verification of Platform Independent Models, Workshop on Model Driven Architecture in the Specification, Implementation and Validation of Object-oriented Embedded Systems (SIVOES-MDA), San Francisco, October 2003.