ETSI TR 103 168 V1.1.1 (2011-02)

Technical Report

Methods for Testing and Specifications (MTS); Application of Model-Based Testing in the Telecoms Domain

Reference

DTR/MTS-00125 MBT

Keywords

application, methodology, report, testing

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

1 Scope

The present document reports on the application of model-based testing in the telecommunication domain. A relevant case study is briefly described in terms of system under test, applied tool chain, together with an overview of the technical requirements. The case study was conducted as part of ITEA2 [i.1] D-MINT project [i.2]. The document concentrates on the results and conclusions from this work, giving an insight into how applicable such methods are today for testing and indicating the current strengths and weaknesses.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ITEA2 web site.

NOTE: Available at http://www.itea2.org; August 2010.

[i.2] D-MINT web site.

NOTE: Available at http://www.d-mint.org; August 2010.

[i.3] Object Management Group; Systems Modeling Language; Version 1.1; November 2008.

[i.4] Object Management Group; Unified Modeling Language (UML) Infrastructure; Version 2.1.2; November 2007.

[i.5] ETSI TS 123 002: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Network architecture (3GPP TS 23.002 version 6.10.0 Release 6)".

[i.6] Heikki Kaaranen, Ari Ahtiainen, Lauri Laitinen, Siamäk Naghian, and Valtteri Niemi, editors. UMTS Networks, 2nd Edition. John Wiley & Sons, Ltd., 2005.

[i.7] Object Management Group; Object Constraint Language (OCL); Version 2.0; May 2006.

[i.8] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In ICSE '99: Proceedings of the 21st international conference on Software engineering, pages 285-294. IEEE Computer Society Press, 1999.

[i.9] Wolfgang Prenninger, Mohammad El-Ramly, and Marc Horstmann. Model-Based Testing of Reactive Systems, chapter Case Studies. Number 3472 in Advance Lectures of Computer Science. Springer, 2005.

[i.10] ETSI TS 124 008: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Mobile radio interface Layer 3 specification; Core network protocols; Stage 3 (3GPP TS 24.008 version 6.10.0 Release 6)".

[i.11] ISO/IEC 9646-1: 1984, Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts.

[i.12] ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)".

[i.13] Malik, Q.A.; Jaaskelainen, A.; Virtanen, H.; Katara, M.; Abbors, F.; Truscan, D.; Lilius, J.; Model-Based Testing Using System vs. Test Models - What Is the Difference?; 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS); 2010; 2010; pages: 291 - 299.

[i.14] Abbors, F.; Backlund, A.; Truscan, D.; MATERA - An Integrated Framework for Model-Based Testing; 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS); 2010; pages: 321 - 328.

[i.15] Abbors Johan; Increasing the Quality of UML Models Used for Automatic Test Generation; Embedded Systems Laboratory, Faculty of Technology, Åbo Akademi University; Master's Thesis; 2009.

[i.16] ITU-T RECOMMENDATION X.680; Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[i.17] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3".

[i.18] Thomas Deiß, Andreas J. Nyberg, Stephan Schulz, Risto Teittinen, Colin Willcock; Industrial Deployment of the TTCN-3 Testing Technology; IEEE Software, July/August 2006, vol. 23, no. 4; pages 48 - 54.

[i.19] OMG Model Driven Architecture Guide; Version 1.0.1; June 2003.

[i.20] OMG Meta-Object Facility (MOF) Core Specification; Version 2.0; 2006.

[i.21] Eclipse Modeling Framework project web page.

NOTE: Available at http://www.eclipse.org/modeling/emf/. August 2010.

[i.22] Tuomas Pääjärvi; Generating Input for a Test Design Tool from UML Design Models; Embedded Systems Laboratory; Faculty of Technology; Åbo Akademi University; Master's Thesis; 2009.

3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ASN.1	Abstract Syntax Notation, number 1
BSS	Base Station Subsystem
EMF	Eclipse Modelling Framework
HLR	Home Location Register
MBT	Model-Based Testing
MDA	Model Driven Architecture
MGW	Media Gateway
MOF	Meta-Object Facility
MSC	Message Sequence Charts
OCL	Object Constraint Language
RNS	Radio Network Subsystem

SDL	System Description Language
SUT	System Under Test
SysML	Systems Modeling Language
TTCN-3	Testing and Test Control Notation version 3
UML	Unified Modeling Language

4 Case study introduction

The case study is introduced by describing three major aspects. First, the System Under Test (SUT) is described in clause 4.1. Second, the case study requirements are elaborated in clause 4.2. Third, the related processes and tools are introduced in clause 4.3.

4.1 System Under Test

The System Under Test in the case study is the Mobile Services Switching Centre Server (MSC Server) of 2nd and 3rd generation mobile networks. An example configuration of the network architecture is depicted in Figure 1. The MSC Server is connected via standardized interfaces to a 2nd generation, GSM, Base Station Subsystem (BSS), a 3rd generation, UMTS, Radio Network Subsystem (RNS), the Home Location Register (HLR) for subscriber data, and the Media Gateway (MGW) transporting the actual user data. The mobile is connected to the BSS or RNS, connection to the MSC Server it using logical links only. The details of the network architecture are specified in TS 123 002 [i.5]. Evolution from 2nd generation GSM systems to 3rd generation UMTS networks and detailed description of the latter technology are provided by Kaaranen et al. in [i.6].

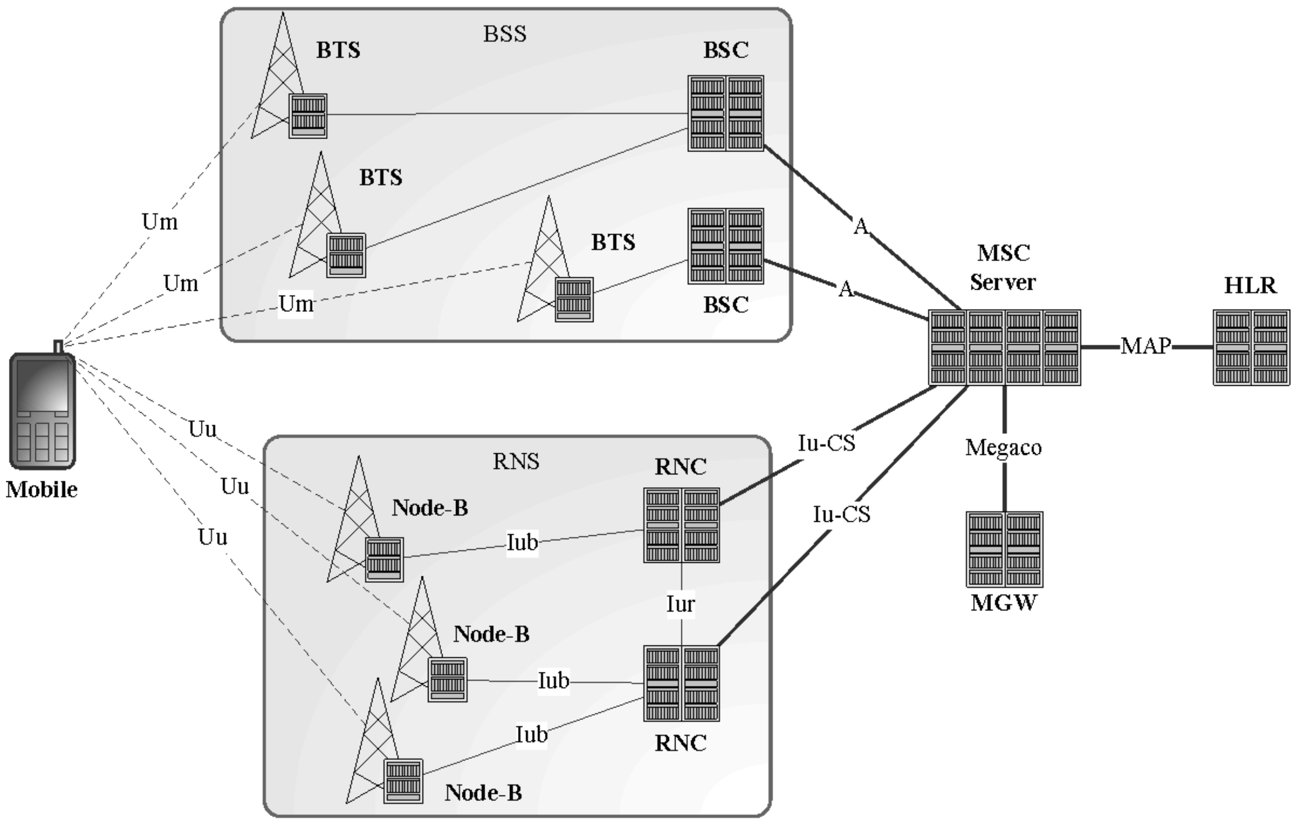

Figure 1: An example configuration of network elements using in the case study

4.2 Requirements

Prior to the case study it was already known that model-based testing was used successfully in several projects in industry. For example Dalal et al. list a number of case studies [i.8] and more recent results are explained by Prenninger et al. in [i.9]. Hence, in this case study focus was on automation aspects on the model-based testing (MBT) and applying MBT in telecommunication domain.

The automation aspects were covered by the process description and the tool-chain implementation described in clause 4.3. This aspect was considered also from legacy perspective, i.e. how MBT can be integrated to the existing processes and tools used in telecom product development.

Many test generation tools used in MBT domain takes *test models* as input. In contrast to such an approach *system models* were chosen to model behaviour of the system in this case study. Difference of these approaches are explained by Malik et al. in [i.13]. The selection was made to exploit system models produced earlier in software development process. In addition, the intention was to investigate are the generated tests meaningful and efficient for product development in telecom domain. Yet another aspect set for the case study was to investigate reuse of existing material available in the telecommunication standards, e.g. use of Message Sequence Charts (MSCs), ASN.1 (Abstract Syntax Notation, number 1) definitions.

In first glance automatic test generation may look like the perfect solution for testing as the test generation is able to produce lots of test cases. However, if there are too many test cases, test generation will take too much time. Therefore, it was a significant evaluation aspect to find out how long test generation takes, how to control the amount of test cases produced by the test generation, and how long it takes to execute the generated test cases.

Finally, use of test generation differs from non-MBT testing due to fact that the generated tests may change significantly after the models are modified and tests are regenerated. In fact, tests may not be comparable between different test generation rounds. This required investigations how to trace progress and coverage of testing.

4.3 Process and tool-chain

The SUT is tested using a process depicted in Figure 2. Four major phases can be identified from the process. First, requirements are processed and described using Systems Modeling Language (SysML) [i.3]. In addition, the system is described using Unified Modeling Language (UML) [i.4] models including references to the SysML requirements. The models are validated using a set of validation rules in order to improve the quality. Second, tests are generated from the models. The test generation phase produces executable test scripts. Third, the test scripts are executed with help of a test execution system. The execution phase produces test logs that are used for further analysis. Fourth, tests are analyzed in case of failures and requirement coverage tracing is performed.

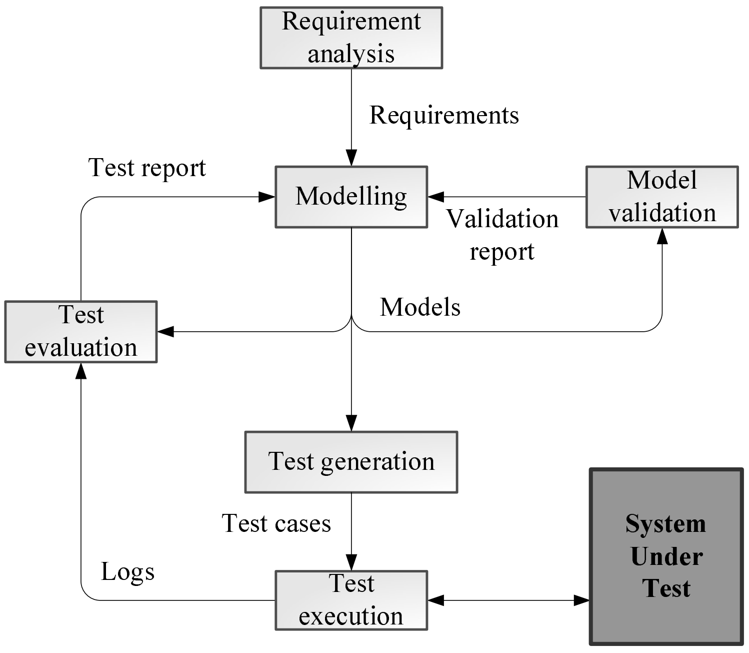

Figure 2: An overview of the case study process

The process is supported by a tool-chain developed in the case study. The tool-chain is depicted in Figure 3. UML models are edited with a model editor. A model validation tool was used to ensure custom rules implemented using Object Constraint Language (OCL) [i.7]. The UML models are transformed with the model validation tool into models in a proprietary UML based modelling language and given as input to the test generation tool. The generation tool produced test scripts that are executed a protocol simulator. The protocol simulator had the role of a test system. It executed test cases by sending and receiving messages, i.e. using asynchronous communication. The test logs produced by the protocol simulator were analyzed and evaluated against the original models using the test evaluation tool [i.14].

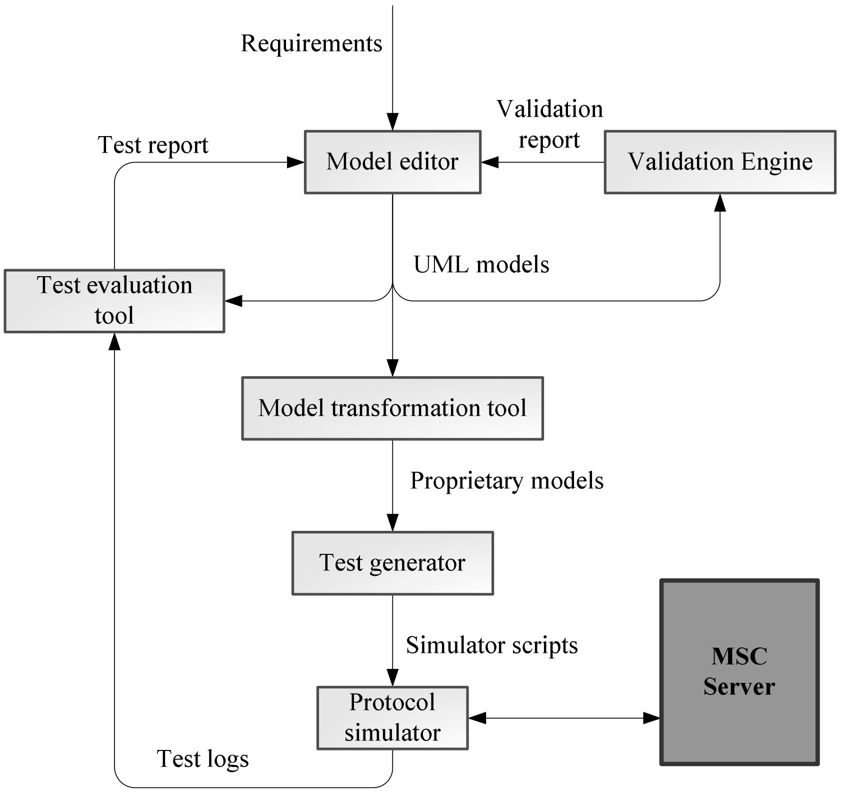

Figure 3: Description of the case study tool-chain

5 Modelling the system

5.1 Modelling competence

Lack of modelling competence prior to the case study did not cause any major issues regarding modelling as such. Typical minor problems have related to the specifics of the proprietary modelling language used by the test generation tool and to Unified Modeling Language (UML). The specific issues are typically related to the complex concurrent behaviour and synchronisation of the parallel state charts which can be difficult to comprehend regardless of the modelling language.

5.1.1 Use of system models

In the beginning of the case study the use of the system models caused problems for the team that had test programming background. The problem was caused by the fact that test programming (and also use of test models) observes the behaviour from the test system point of view instead of the system's point of view. However, when using the system models, the point of view is the opposite as explained in clause 4.2. For example, when a mobile should send LOCATION UPDATING REQUEST message to the MSC Server and the MSC Server should respond with LOCATION UPDATING ACCEPT message [i.10]. When observing the behaviour from test system (that is from the mobile) point of view, the test system uses following procedure.

1) Send LOCATION UPDATING REQUEST
2) Receive LOCATION UPDATING ACCEPT

However, when modelling the behaviour of the system (that is the MSC Server), the sequence is following. Note that although the order of the messages is the same, the operations are the exact opposite.

1) Receive LOCATION UPDATING REQUEST
2) Send LOCATION UPDATING ACCEPT

This logical aspect may look fairly insignificant but it takes time to adapt to this new way of thinking if one has had test programming background. For the programmers who have used to look from the SUT's point of view, this should not be an issue. At the end, it is easy to find such logical mistakes when test cases are executed.

For industrial use, describing only the behaviour of the SUT is not sufficient. For example, when testing network elements of a telecom network it is necessary to describe the relation of the SUT and other network elements, i.e. the environment of the SUT. For this purpose, additional models have been used to describe the network architecture and the configurations of the architecture for the tests. Such models are not system models nor test models as such because of the models describe both the system under test and the surrounding network. This should be taken into account when talking with MBT enthuastics who may debate among themselves on the pros and the cons of system models vs. test models.

5.2 Tool support

Model development can be significantly improved but also hindered by model editors. Obstructions may arise when the modelling tool has a lot of features and the features are difficult to use. In such cases modelling slows down due to poor usability. In fact, this might slow down modelling significantly and lead to false conclusions on the modelling. However, a simple editor is not sufficient either for professional modelling because of eventually models will grow, become complex and require maintenance. Hence, the usability is an important factor when choosing model editor.

During the case study few modelling tools were used but thorough comparison of the tools were not made. The most simplistic editor was provided for free as part of the test generation tool to get started with modelling. It is a light-weight state-chart drawing tool that lacks model maintenance features. Hence, it is not sufficient for professional modeling. The rest of the used tools were fairly complex in terms of features and could not be used efficiently without guidance. User manuals and on-line helps are helpful in case of some editors. In case of one of the editors the basic modelling did not require any documentation and the tool works fine. However, it is expected that the documentation will be thorough enough for unexperienced users when more advanced features are needed for modelling.

In this case study, the quality of models was assurred with a help of a number of different tools of the tool-chain. The model editors provide basic syntax checking. In addition, a set of OCL rules were used perform static semantic analysis on the models as described in [i.15]. Also, the tool-chain supports to discover suspicious model constructs, e.g. reachability of the states is checked by the test generation tool and incomplete parts of the model are checked by model editors or by the test generation tool.

Various telecommunication specifications exploit ASN.1 notation standardised in ITU-T in X series specifications (for basic notation see [i.16]). ASN.1 is used to define types and procedures of protocols exchanged on interfaces of network elements. ASN.1 type and procedure definitions are detailed enough for code generation. Hence, it would be beneficial for relevant testing project as well to reuse directly the ASN.1 type definitions. However, currently ASN.1 types are not supported directly which is a drawback compared to for example TTCN-3 language which has built-in support for ASN.1 type definitions [i.17].

6 Test generation

Test generation is applied for offline mode, i.e. a test is first generated and only after that the test can be executed in contrast to online mode where test generation and execution is interleaved by computing a step of a test and execute it instantly before performing any actions for the next step. Offline mode has also an advantage of re-executing test cases without a need to run the test generation phase again e.g. for regression testing purposes.

Test generation was performed by using a test generation tool that offered a plug-in for scripts used by the protocol simulator. The test generation is done in two phases. First, the test generation tool derives test cases from models specified in its own proprietary UML based modelling language. Second, the plug-in rendered the test cases into protocol simulator script format.

The test generation can be controlled using test generator's configuration parameters for algorithmic configuration, requirement coverage configuration, and state machine coverage configuration. Test generation parameters can be selected individually. Because of this freedom it is easy to define a configuration that leads to very long test generation times. Also test generation times depends on the size and the complexity of the models. However, the tool did not perform full generation when the model is modified but only for the changed parts of the model. This reduced the test generation times after the tests are generated first time.

At the time of the case study, the test generation tool did not support non-deterministic models in offline mode. This is an issue for complex telecom systems that may behave in non-deterministic manner due to a combination of several factors (e.g. concurrency, network element internal computation tasks, load balancing) affecting to the behaviour of the SUT. Clearly, for complex systems like the MSC Server, this is not sufficient situation. Technically speaking, it is possible to get around of this problem by handling non-deterministic cases at the adaptation level of the test system instead of the models. However, this leads to increased effort and complexity that should be avoided.

7 Test execution

Model-based testing using offline mode has not imposed any major technical changes for test execution tools compared to non-MBT approaches. For example, ISO/IEC abstract testing methodology given in [i.11] and general structure of a TTCN-3 test system described in [i.12] can be still used. This is in fact a good news from legacy test system point of view. It is not necessary to make major modifications for the legacy test systems. Hence, MBT does not create additional costs for test execution phase.

However, MBT has impact on how the tools are used due to fact that requirements are tracked consistently throughout the process including test execution phase. Requirement tracking required injection of requirement identifiers into test cases and test logs as illustrated in Figure 4. The requirement identifier is denoted as "Req ID" in the figure.

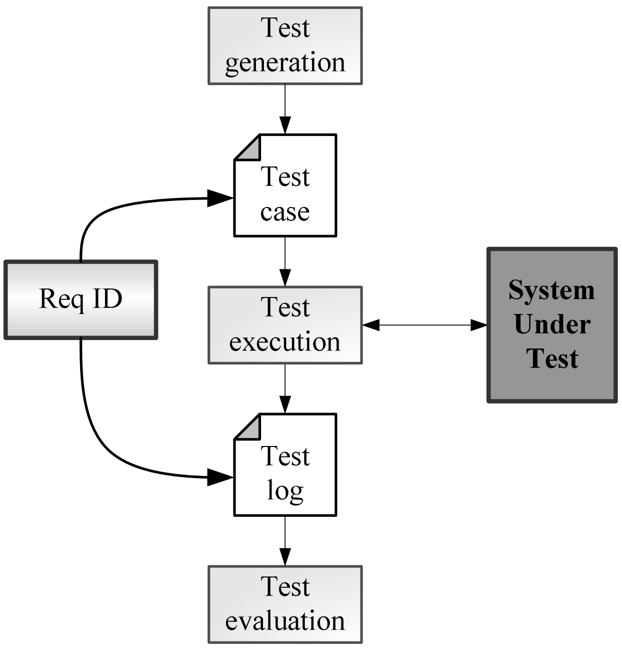

Figure 4: Embedded requirement identities in test cases and test logs

8 Test analysis

Test analysis refers to a phase that is done after a set of test cases are executed in order to investigate the results of the test executions.

The test analysis phase reports what has been covered by executed test cases. This is achieved by using the information embedded into the test logs. For example, in the case of requirement coverage requirement identifiers are injected to generated test cases and during the test execution the identifiers are explicitly recorded in to test logs as depicted in Figure 4.

For the succeeded tests, a report is created to provide an overview of the test execution. Also the report contains information on the coverage.

For the failed test cases, a set of OCL constraints are produced based on the test execution logs. These constraints can be used by the model editor to express explicitly which parts of the UML models do not correspond to the behaviour of the SUT. Although this does not provide a pointer to the root cause of the failure, it automates some of the manual tasks typically done in test result analysis phase. In addition, also for the failed test cases the report indicates which of the coverage goals are not reached.

9 Tool integration

Tool integration aspects focus on how the functionality implemented by individual tools or prototypes were integrated in the overall tool-chain. Findings are provided in following clauses.

9.1 Lack of common exchange format

A prerequisite for any tool is that it can be integrated into the existing tool-chain. The less work is required for the integration, the more likely it is that the tool will be deployed in large scale. A practical but an unfortunate consequence of this fact is that a tool with sophisticated and powerful features may be discarded if it is difficult to understand and learn how integration can be done.

In this case study, the integration of the tools has been an issue that has required a lot of attention. In particular, the implementation and the integration of the SUT adapter have required a lot of effort. Although the SUT adapter does not have a significant role from a model-based testing point of view, it is an essential entity that supports the communication with the system under test. Hence, it is unfortunate that a lot of effort has been spent on the SUT adapter that has little to do with model-based testing as such. In the end, this issue is not significantly different from similar tool-chains regardless whether model-based technology has been used or not. In fact, this is in-line with our previous experiences with test tools and test tool-chains, e.g. on TTCN-3 experiences reported in [i.18].

In this case study, similar to many other cases, the problem is solved by implementing a number of adapters to integrate tools. However, the use of adapters has few drawbacks. Implementation of adapters requires extra work and the adapter adds complexity of the tool-chain. The first issue increases cost of tool development and the second issue reduces the performance of the tool-chain.

A part of the problem is to deal with legacy test tools that may have a history of years or even decades. These tools cannot be ignored and hence these tools need special attention. Most of these tools can be integrated to the tool-chain by implementing adapters or by using transformations. However, it is somewhat surprising that there is no widely used de-facto or standardised tool integration frameworks as this issue has been around for years.

In the context of model-based testing there are few well-known approaches, such as OMG's Model Driven Architecture (MDA) [i.19] and related standards like UML and Meta-Object Facility (MOF) [i.20]. In addition, Eclipse Modelling Framework (EMF) [i.21] is a Java framework that supports modelling that utilizes the MOF. Although OMG's MDA and Eclipse Modelling Framework provide frameworks that support information exchange between tools, readily available integration framework is often missing.

Based on the experience in the past and also on this case study, one of the findings is that if the framework is complex and difficult to comprehend, preferably other alternatives will be chosen. Eclipse is fairly good example of this. Eclipse web site provides a lot of information on Eclipse framework but for some one who is not familiar with Eclipse it is difficult to comprehend how things provided by Eclipse projects could be exploited. In the end, it is difficult to filter out from the mass of information what details are relevant and what are not. Although Eclipse is used here as an example, similar experiences can be pointed out e.g. with MDA and related standards. It is fair to say that it requires a lot of expertise to exploit EMF and MDA in an efficient and feasible way. If the initial learning effort could be reduced, it would be more likely that such frameworks would be used more often and possibly become more popular integration frameworks.

9.2 On-line mode

At the time of the case study was performed the use of the on-line mode in the case study was not investigated in-depth. The reason for not including the on-line mode was the runtime performance of the model-based testing tool that did not meet the real-time requirements set by the SUT.

However, initial attempts to model exploited on-line mode and based on experiences on the early stage mock-up modelling it was considered that the on-line mode may be beneficial in context that do not have strict real-time requirements. The major advantage the on-line mode offers is caused by its ability to provide rapid feedback between model development and execution.

9.3 Offline mode

The current tool-chain with off-line mode has been demonstrated in several occasions and it is proven to work in practise. Currently the off-line mode consists of model validation and test generation tools and a protocol stack simulator. The model validation tool is used to perform UML to proprietary language transformation [i.22]. From the generated models the test generation tool generates tests. The generation is done using a special plug-in that outputs scripts for the protocol simulator. The produced scripts can be executed with the protocol simulator which also provides the SUT adapter functionality. The model validation tool can also be used to trace requirements from models to test scripts and also to test logs, and back-trace requirements from test scripts to UML models of the SUT.

Several test cases have been executed using the current tool-chain. Although the tool-chain works well it still requires more work to satisfy all the original requirements. For example, the existing setup lacks support for statistical testing and ability to pay special attention to functionality that has revealed most of the faults. Although, these features can be considered additional features, the features are important in order to convince the users to swap their existing tool set with a tool set that provides more advanced features.

10 Special issues

Special issues clause describes issues that are not related to any the process or the tools described in the present document. Instead, the special issues list topics that are general by nature.

10.1 Model-based testing and telecommunication standards

Various telecommunication specifications exploit ASN.1 notation standardised in ITU-T X series (for basic syntax see [i.16]). ASN.1 is used to define types and procedures of protocols exchanged on interfaces of network elements. ASN.1 type and procedure definitions are detailed enough for code generation. Hence, it would be beneficial for relevant testing project as well to reuse directly the ASN.1 type definitions. However, use of ASN.1 definitions cannot be used as part of UML models which is a drawback compared to for example TTCN-3 language [i.17]. Similar problems exist with other languages and notations. For example, use of System Description Language (SDL) descriptions cannot be reused as such and message sequence charts embedded as figures need to be redrawn using UML as the available message sequence charts are not in a machine processable format.

Although standards are primarily abstract descriptions instead of concrete descriptions, reuse of the definitions that are parts of standards, would speed up development of the products that are based on the standards.

Obviously, tool providers can influence on this issue by providing transformations between the languages and the formats. However, currently it seems that there is no consensus and no de-facto or standardised exchange format exists.

Naturally this requires balancing between the standardisation organisations, tool vendors and customers' demands.

10.2 Coverage

Test scripts written manually are fairly static by nature in terms of the purpose of the test cases. Once a test script is written its main purpose remains more or less the same. After a while test engineers become familiar with the test script names and know immediately what the test script is all about.

In case of test generation the situation is different. There are no guarantees that the content of test cases will remain the same for each test generation rounds, i.e. whenever test generation is performed, the content of test cases may change completely.

Therefore, measuring test coverage by listing a set of test cases does not provide meaningful results. Instead, other approaches have to be used. One approach is to use techniques that provide coverage information on the model structure, e.g. state coverage, transition coverage, branch coverage. Although these techniques provide systematic and valuable information on test coverage, other approaches are needed to be more precise on the what the model suppose to do. An approach to deal with this aspect is to embed requirements or other information that can be tracked throughout the process and the tool-chain. If the requirements are detailed enough and express the product requirements, then the requirement coverage can be achieved and worthwhile of considering.

History

Document history		
V1.1.1	February 2011	Publication