MATERA - An Integrated Framework for Model-Based Testing

Fredrik Abbors, Andreas Bäcklund, and Dragoş Truşcan
Department of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520, Turku, Finland
Email: {Fredrik.Abbors, Andreas.C.Backlund, Dragos.Truscan}@abo.fi

***Abstract*—This paper presents MATERA, a framework that integrates modeling in the Unified Modeling Language (UML), with requirement traceability across a model-based testing (MBT) process. The Graphical User Interface (GUI) of MATERA is implemented as a plug-in in the NoMagic's MagicDraw modeling tool, combining existing capabilities of MagicDraw with custom ones. MATERA supports graphical specification of the requirements using SysML and tracing of them to the UML models specifying the SUT. Model validation is performed in MagicDraw using both predefined and custom validation rules. The resulting models are automatically transformed into input for the Conformiq Qtronic tool, used for automated test generation. Upon executing the test scripts generated by Qtronic in the NetHawk's East execution environment, the results of statistic analysis of the test run are presented in the GUI. The back-traceability of the covered requirements from test to models is also provided in the GUI to facilitate the identification of the source of possible errors in the models. The approach we present shows that existing model-based languages and tools are an enabler for model-based testing and for providing integrated tool support across the MBT process.**

***Keywords*-Model-Based Testing; Model Validation; Requirements Traceability;**

I. Introduction

The software industry is facing several challenges in delivering increased business value to their customers. Customers demand more flexible, reliable, low cost, and highly efficient software solutions. Additionally, the customers usually require the software systems to be deployed within different types of distributed environments. Research has shown that as much as 60 per cent of the total development time can be spent in testing [1]. This implies that testing is a highly expensive and time consuming process.

Model-based testing is a technique that tries to address these issues by introducing automatic generation of tests from models representing the behavior of the System Under Test (SUT). Using models for test generation increases the pressure put on the modeling process and on the quality of the models used for test generation, as any inconsistency in the models will reflect later on in the quality of the generated test cases.

In order to address these issues we suggested a modeling approach [2] which puts emphasis on three aspects. First, the models of the SUT are built in a systematic manner starting from requirements, after which they are fed as input to the test generation tools. Secondly, several model types are used to model different perspectives of the system like behavior, data, architecture, test configuration. Thirdly, the requirements of the SUT are traced through all the stages of the testing process and back-traced from the executed test cases back to models. The approach and the afferent tool support is also referred to as MATERA (Modeling for Automated TEst deRivation at bo Akademi).

In this paper, we describe how the tool support for the MATERA approach is provided by reusing and adapting existing commercial tools. More specifically, we try to show that existing model-based techniques and tool can become an enabler for supporting model-based testing and for providing integration across the MBT tool chain. MATERA has been targeted to the telecommunications domain and thus the examples used in this paper are excerpts from and industrial case study. Further information on case study can be found in [3].

II. MATERA

The MATERA tool-set is used to provide support for the approach by integrating modeling in the Unified Modeling Language (UML) [4] and requirement traceability across a custom MBT process (see Figure 1). As mentioned in the introduction, a set of models are created from the system requirements. These models are validated by checking that they are consistent and that all the information required by the modeling process is included. Consequently, the models are transformed into input for the test derivation tool. The resulting test cases are executed (after being implemented) using a test execution framework. The results of the test execution are analyzed and a report is generated. Requirements are linked to artifacts at different levels of the testing process and finally attached to generated test cases. This allows one to traceback to models which test cases have covered different modeling artifacts or from which part of the models a failed test case has originated.

In the following, we briefly present different features of the MATERA tool set accompanied by excerpts from a telecom case study.

A. Graphical User Interface

The GUI of the MATERA tool-set has been implemented as a plug-in in the MagicDraw UML modeling tool [5]. The plug-in is developed in Python using the Open Application

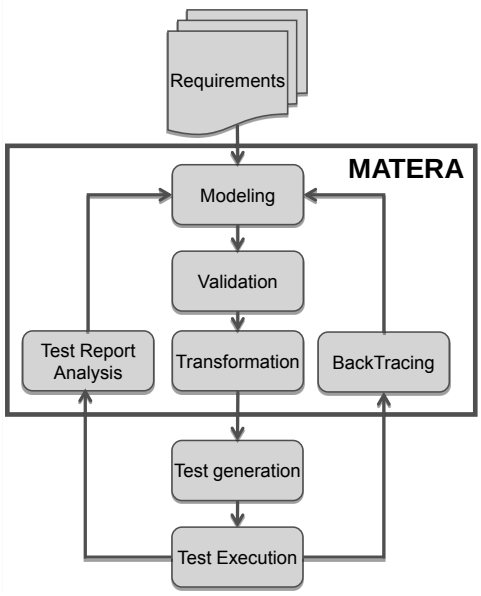


Figure 1. MATERA process

Programming Interface (Open API) of MagicDraw. The purpose of MATERA tool-set is to extend the capabilities of MagicDraw for specifying system models and using them as input for automatic test generation. Once the models are completely specified, they can be transformed to input for test generation tools. Besides model transformation, MATERA also supports model validation, test reporting, and (back-)tracing of requirements. Hence, MATERA promotes the integration of UML modeling with test generation tools. Figure 2 shows a caption of the MATERA GUI in MagicDraw.

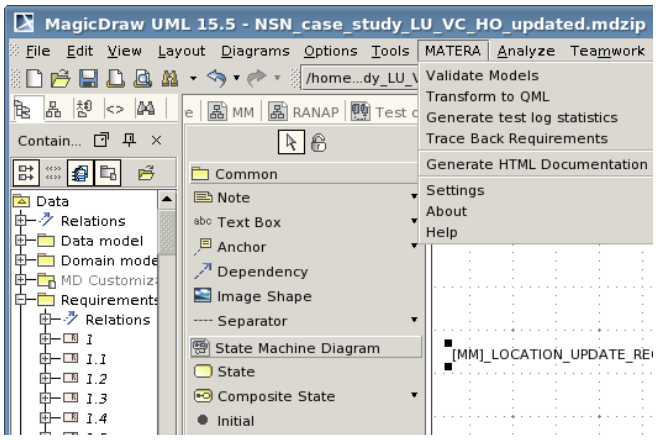


Figure 2. Caption of the MATERA GUI in MagicDraw.

B. Requirements Modeling

Requirements play an important role in any software project and it is also the starting point of the testing process [6]. MATERA starts with the analysis and structuring of the informal requirements into a Requirements Model. The Requirements Diagrams of the Systems Modeling Language (SysML) [7] are used for this purpose. We use MagicDraw's model editor to create, edit, and structure requirement elements. Requirements are organized hierarchically in a tree-like structure, starting from top-level abstract requirements down to concrete testable requirements. Figure 3 shows how requirements are structured in MagicDraw editor. Each requirement is described using a *Name* and an *Id*, a *Text* field explains the requirement, and a *Source* field points to the document or standard from which the requirement was extracted. Further, requirements can be related to each other using the relationships defined by SysML. For instance, requirements can be derived into other requirements using the *deriveReqt* relationship or related horizontally using the *trace* relationship.

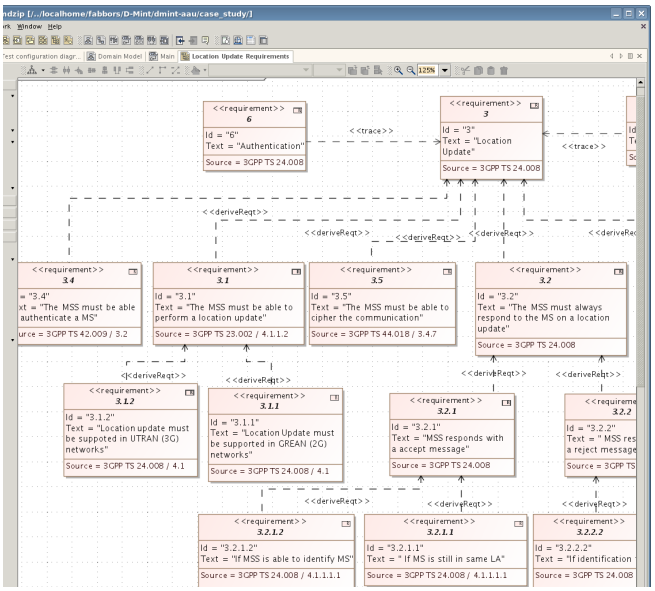


Figure 3. Structuring requirements in MagicDraw.

Traceability of requirements is a pivotal aspect of MBT that allows one to ensure that all requirements have been tested [8]. As the models are derived from requirements, it is important to track how different requirements are reflected in the models, on different perspectives, and on different abstraction levels. In MATERA, the requirements can be linked to different parts of the UML-based system specification, for instance to models or to model elements, to ensure requirements traceability throughout the process. When the specification is used for test generation, the requirements are associated with the generated test cases and propagated throughout test execution.

With MATERA it is possible to check that all specified requirements have been properly linked to models or model elements. For this purpose we use MagicDraw's validation

engine and custom Object Constraint Language (OCL) [9] rules to check e.g. that the leaf requirements are not left unlinked or that every requirement has a unique Id.

C. Modeling the SUT

In MATERA, we take advantage of the expressiveness and graphical capabilities of UML for creating the specification of the SUT. In our case the test model is derived from high-level development models, such that partial reuse of the development specification is enabled. In addition, the SUT is specified from several perspectives to enable a successful test derivation process. These perspectives of the SUT are modeled using the UML diagram editors provided by MagicDraw; a class diagram is used to specify an architectural model describing the static structure of system. The architectural model shows what domain components exist and how they are interrelated through interfaces. A behavioral model describes the dynamic behavior of the SUT using state machines. Data models are represented as class diagrams and are used to describe the data exchanged between different domain entities. Last but not least, test configuration models, represented as object diagrams, are used to describe specific test configurations and to set up initial values for the test components.

In MATERA, different parts of the specifications are linked together in order to specify dependencies. We use MagicDraw's property editor to link model elements and data together, for example, every messages specified on an interface in the domain model is linked to the corresponded class in the data model describing the structure of the message. Also the properties of the elements can be edited using the Specification editor, see Figure 4.

D. Model Validation

Humans tend to make mistakes and forget things. Therefore, to gain efficiency of using a MBT process and reducing the costs by discovering faults at an early stage, it is necessary to validate the models before using them to e.g. automatically generate code or test cases [8]. Hence, a set of modeling guidelines and validation rules have been defined for ensuring the quality of the models. Modeling guidelines are used to specify how different models are created from requirements or from other models, what information they should contain, how this information is related to the information present in the other models, etc.

Validation rules have been defined and implemented for both Requirements Models and for System Models for checking different quality metrics of the resulting models before proceeding to the test derivation phase. These rules ensure that the models are syntactically correct, they are consistent with each other, and that they contain the information needed in the later phases of the testing process. In MATERA, validation is prerequisite before transforming the models.

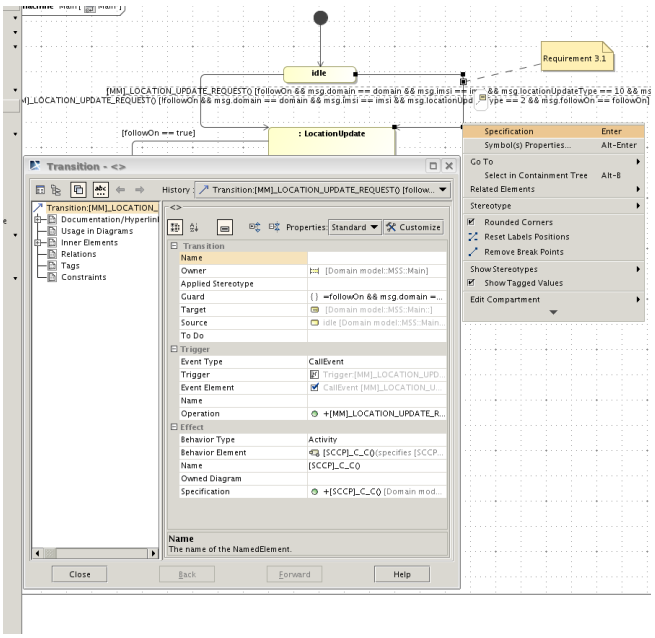


Figure 4. Screenshot showing the specification editor of a transition.

OCL is used to describe rules that apply to models. The rules describe conditions that must hold for the system being modeled. MagicDraw comes shipped with a set of predefined OCL rules for validating UML and SysML models. In addition to those, custom rules have been defined and implemented specifically for MATERA. The custom rules we created are all related to the modeling process, the application domain, and the specific MBT tool we target. For instance, we have created validation rules for checking the leaf requirements in the Requirements Diagram are linked to model elements and that messages defined on interfaces in the domain model are linked to data models.

An OCL rule is similar to a model element which has a number of editable properties e.g. name, specification, constrained element, severity level, etc. In order to provide reuse, rules are stored in different validation suites (packages) depending on the intended purpose of the rule, see Figure 6.

MagicDraw has a built-in validation engine for checking the rules against models. The validation in MagicDraw can be invoked at any time. When the validation is started the user will be prompted for the validation suite that he/she wants to apply, the scope (which models), and the severity level. Upon running the selected validation suite in the validation engine, MagicDraw creates a summary of the validation process as depicted in Figure 5, listing which elements are violating a rule and why. From this window the user can e.g. choose to open all diagrams with the elements violating a rule and see the faulty elements in the diagrams as they are highlighted. Once an error has been corrected the user can run the same validation suite again to see if the

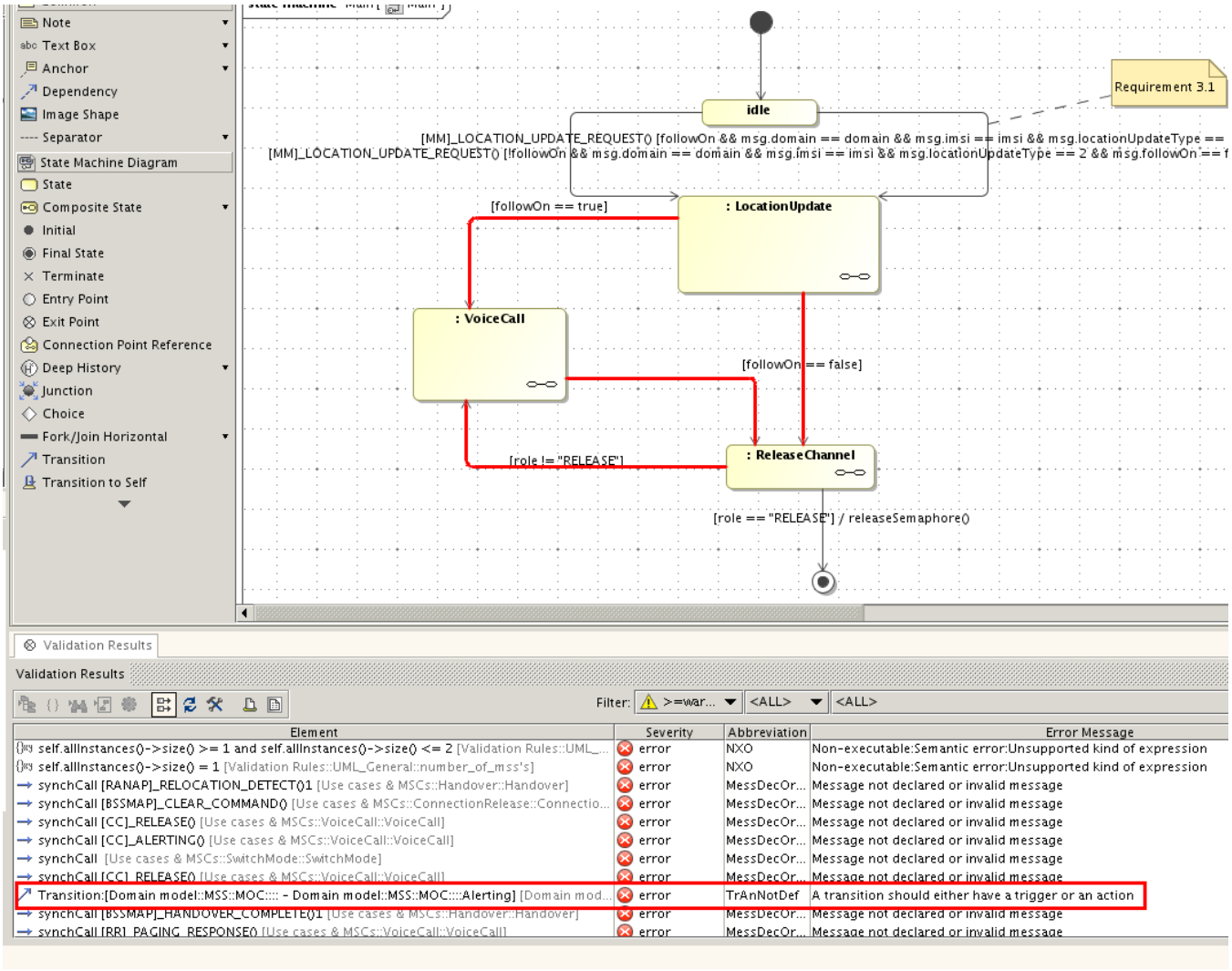


Figure 5. Validation summary.

modifications made any difference.

E. Transformation

In MATERA, the system models of SUT are transformed into input for the automated test derivation process. The transformation has two steps. First, the needed information from the models is collected by a parser module and stored into an internal representation. Then this information is read, by various build modules, and written (rendered) in the format supported by the test generation tool. The idea is to have a generic transformation approach and to be able to expand the approach to target different test generation tools. However, in our research we currently target only one particular test generation tool, namely Conformiq's Qtronic [10]. The transformation [11] [12] translates UML models to the Qtronic Modeling Language (QML), the language used by Qtronic for specifying the SUT.

The transformation also propagates requirement from UML models to QML. In QML, requirements are treated as textual tags attached to different parts of the specification, which are treating as testing goals during the test generations process. Once the test cases are generated they are implemented in the language used by the test execution tool (NetHawk's EAST [13] in our case) using the a scripting backend. During this process the requirements are propagated further and attached to executable tests allowing the test execution tool to trace and log the execution of tests cases and their associated requirements.

F. Test Reporting

MATERA offers support for test reporting. The test report will summarize the result of the testing process in terms of generated test cases, verdicts, coverage levels, etc. The information in the test report is collected form test logs and

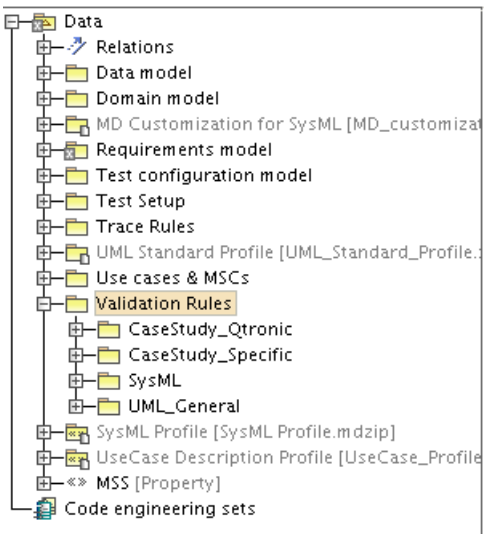


Figure 6. Validation suites in MATERA.

from the test scripts by comparing the test purposes encoded in the scripts against the results of the test execution.

When invoking the test report function from the MATERA menu (see Figure 2), a parser module collects and stores data from the test logs and scripts, similarly to the transformation. The collected data is then analyzed and presented to the user in HTML format, using the systems default HTML reader. The user only has to specify is the paths to the executed test scripts and their corresponding test logs. Figure 7 shows a snapshot of a test report.

G. Back-Tracing of Requirements

In MATERA, information from test logs is collected on how different requirements have been covered during both test generation and test execution phase, respectively. Based on this information the requirements are tracked back to the specifications from which the corresponding test cases have been generated, in order to detect the source of possible faults in the specifications. Upon selecting the *Trace Back Requirements* in the MATERA's GUI a Python script that analyzes the test logs and generates OCL queries that we use in MagicDraw to locate erroneous parts in the UML system models. The OCL queries are used to trace requirements from tests to the requirements models and to the requirements placed on transitions in state machines. The Python scripts generates OCL queries based on the information in the test logs and writes them to MagicDraw in a validation suite called "Trace Rules". We use again MagicDraw's OCL interpreter to find the requirements in the UML models based on the produced OCL queries.

This way one can see which requirements failed during testing and to what model elements they are linked. It also enables one to identify which parts of the system model have been covered by the test set. The back-tracing function in MATERA will highlight model elements in the system models, to which a failed requirement was linked. In Figure 8, the list of requirements covered by failed test cases in presented at the bottom of the screen. By selecting an entry in the list the corresponding requirement is highlighted in the diagram editor. Ultimately, since requirements are linked to model elements, it facilitates the identification of those parts of the models that are not in sync with the SUT, see Figure 9.

III. Related Work

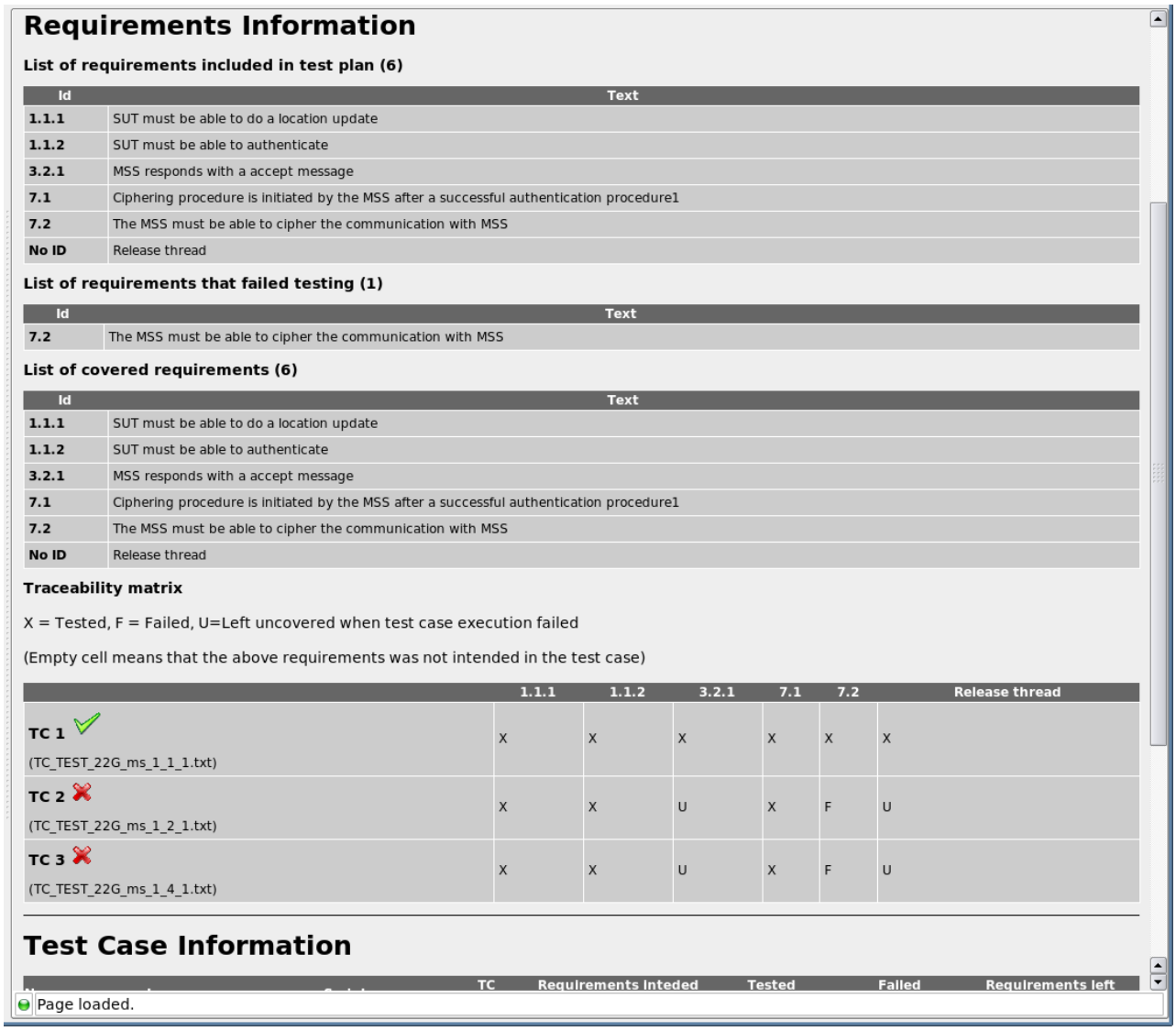
Similar research has been conducted within other industries. When modeling for automatic test generation, it is has proven beneficial to check the models against pre defined modeling rules and design guidelines before generating tests. In [14], the authors use the Object Constraint Language (OCL) to specify design guidelines and modeling rules for Simulink models. This approach is similar to ours, in the sense that rules written in the OCL language are used to check model consistency against a metamodel. However, their approach differs slightly from ours since the authors check the rules against a custom made Matlab/Simulink metamodel while we check OCL rules against the UML metamodel.

Other research similar to ours is described in [15]. In this research the authors use Matlab/Simulink behavioral models, instead of UML behavioral models, from where they automatically generate test sequences. Their approach differs somewhat from ours since it does not address automatic evaluation of test results.

From other reviewed works, the approach presented in [16] is closest to our approach. In there, the authors use a restricted set of UML diagrams together with OCL to describe both the static structure and dynamic behavior of the SUT. Requirements traceability is addressed by manually tagging the UML specification with ad-hoc comment symbols to associate a requirement with an OCL statement. Using the LEIRIOS (now Smartesting) Test Designer tool the authors automatically generate test cases out of the UML system specification and a traceability matrix is obtained after test execution. However, the Test Designer tool does not offer support for tracing requirements from test cases to the UML specification.

IV. Conclusion

This paper has presented a framework for integrating UML and requirements traceability in a MBT process. The MATERA framework is implemented as a plug-in for MagicDraw, which adds extended functionality to use UML models for automatic test generation. UML modeling is combined in MATERA with consistency checking of the specification using pre- and custom defined consistency rules, with the purpose of increasing the quality of the



Requirements Information

List of requirements included in test plan (6)

Id	Text
1.1.1	SUT must be able to do a location update
1.1.2	SUT must be able to authenticate
3.2.1	MSS responds with a accept message
7.1	Ciphering procedure is initiated by the MSS after a successful authentication procedure1
7.2	The MSS must be able to cipher the communication with MSS
No ID	Release thread

List of requirements that failed testing (1)

Id	Text
7.2	The MSS must be able to cipher the communication with MSS

List of covered requirements (6)

Id	Text
1.1.1	SUT must be able to do a location update
1.1.2	SUT must be able to authenticate
3.2.1	MSS responds with a accept message
7.1	Ciphering procedure is initiated by the MSS after a successful authentication procedure1
7.2	The MSS must be able to cipher the communication with MSS
No ID	Release thread

Traceability matrix

X = Tested, F = Failed, U=Left uncovered when test case execution failed

(Empty cell means that the above requirements was not intended in the test case)

	1.1.1	1.1.2	3.2.1	7.1	7.2	Release thread
TC 1 (TC_TEST_22G_ms_1_1_1.txt)	X	X	X	X	X	X
TC 2 (TC_TEST_22G_ms_1_2_1.txt)	X	X	U	X	F	U
TC 3 (TC_TEST_22G_ms_1_4_1.txt)	X	X	U	X	F	U

Test Case Information

TC	Requirements inteded	Tested	Failed	Requirements left

Page loaded.

Figure 7. Caption of a test report generated from MATERA.

specifications used for automated test generation. Requirements are traced across the entire testing process; from models to test cases, and from test cases back to models. Requirements traceability is facilitated in MATERA by the back-traceability function.

The MATERA framework provides value by allowing testers to generate input for automatic test generation tools from a graphical representation of the SUT. The graphical representation is created using UML, which is one of the commonly used standard in the software industry. Additionally, back-tracing of requirements allows for having a visual overview of requirements that have not been properly tested. MATERA also provides the tester with a test report presented in HTML format which contains statistics of the test execution.

In our current research, we have focused mainly on generating input for Conformiq's Qtronic test generation tool. However, in the future we plan to target other test generation tools as well. Another future goal is to include statistical information into the UML models, based on past test executions, to be able to prioritize test cases or to focus the testing on specific parts of the system specification. We will also investigate how the information contained in the UML models can be used for generating adapter frameworks for different MBT tools.

ACKNOWLEDGMENT

Financial support from Tekes under the ITEA2 D-Mint project is gratefully acknowledged.

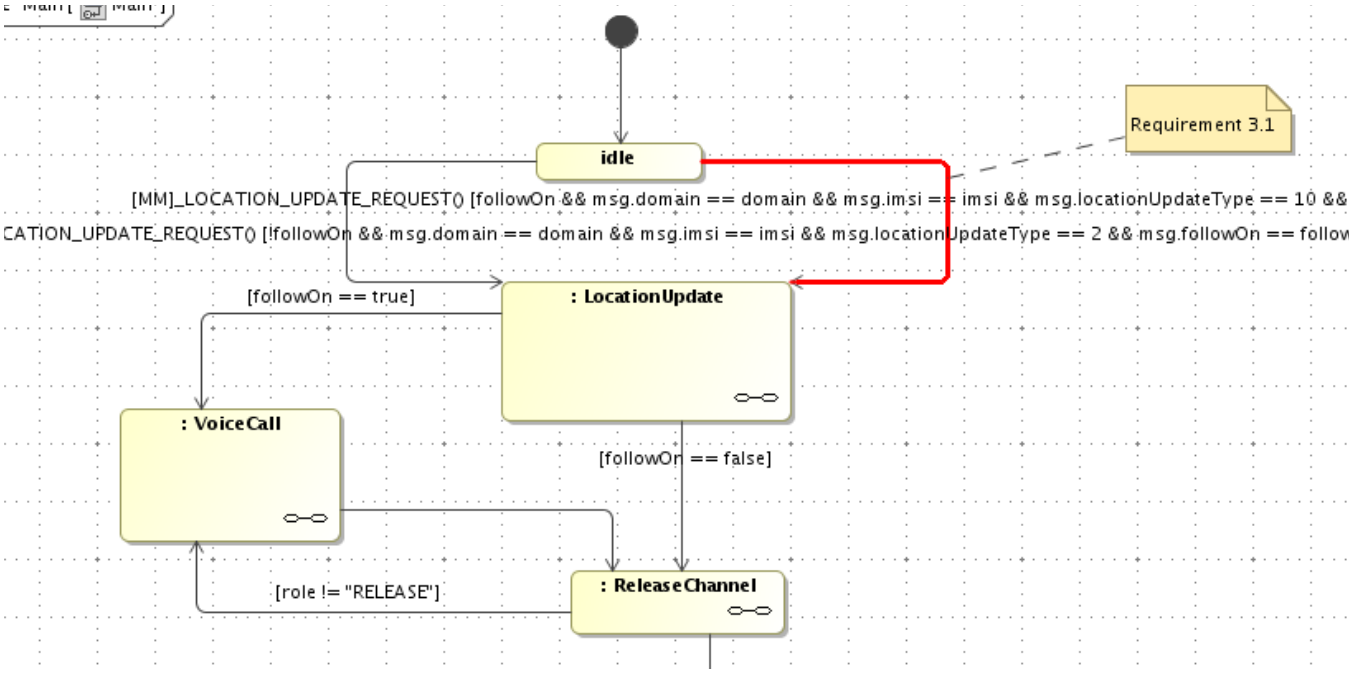


Figure 8. Back-tracing of a requirement to a transition.

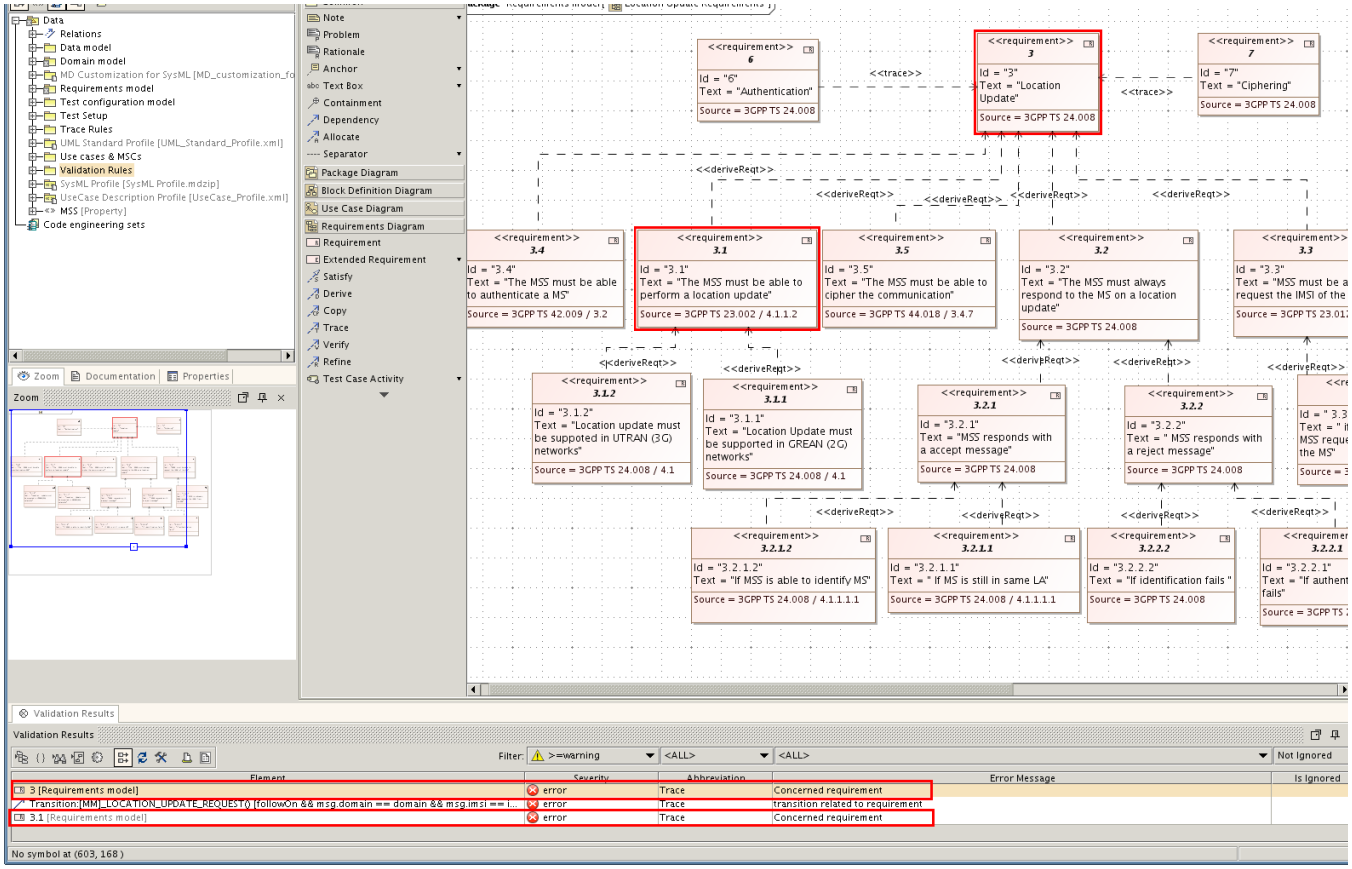


Figure 9. Back-tracing of requirements to the Requirement Diagram

.

REFERENCES

[1] "Historical Perspective in Optimising Software Testing Efforts - http://www.indianmba.com/Faculty_Column/FC139/fc139.html." [Online]. Available: http://www.indianmba.com/Faculty_Column/FC139/fc139.html

[2] J. Abbors, "Increasing Quality of UML Models Used for Automatic Test Generation," Master's thesis, Åbo Akademi University, 2009.

[3] F. Abbors, "An Approach for Tracing Functional Requirements in Model-Based Testing," Master's thesis, Åbo Akademi University, 2009.

[4] "Unified Modeling Language - http://www.omg.org/spec/UML/2.0/." [Online]. Available: http://www.omg.org/spec/UML/2.0/

[5] "NoMagic MagicDraw," http://www.magicdraw.com/.

[6] G. Fournier, *Essential Testing: A Use Case Driven Approach*. BookSurge Publishing, 2007, pp. 67–75.

[7] Object Management Group, "OMG SysML Specification," Tech. Rep. [Online]. Available: http://www.omg.org/spec/SysML/1.1/

[8] M. Utting, *The Role of Model-Based Testing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 510–517.

[9] *Object Constraint Language v2.0*, OMG, May 2006, http://www.omg.org/spec/OCL/2.0/PDF.

[10] "Conformiq Qtronic," http://www.conformiq.com/.

[11] T. Pääjärvi, "Generation Input for the Test Generator Tool from UML Design Models," Master's thesis, Åbo Akademi University, 2009.

[12] F. Abbors, T. Pääjärvi, R. Teittinen, D. Truşcan, and J. Lilius, "A Semantic Transformation from UML Models to Input for the Qtronic Test Design Tool," Turku Centre for Computer Science (TUCS), Tech. Rep. 942, 2009.

[13] NetHawk, "NetHawk EAST," 2009. [Online]. Available: www.nethawk.fi/products/nethawk_simulators/nethawk_ims_tester/

[14] T. Farkas, C. Hein, and T. Ritter, "Automatic Evaluation of Modeling Rules and Design Guidelines," in *proc. of the Workshop "From code centric to Model centric Soft. Eng.", http://www.esi.es/modelware/c2m/papers.php*, 2006.

[15] M. Conrad, H. Dörr, I. Stürmer, and A. Schürr, "Graph Transformations for Model-based Testing," *GI-Lecture Notes in Informatics, P-12*, pp. 39–50, 2002.

[16] E. Bernard and B. Legeard, "Requirements Traceability in the Model-Based Testing Process," in *Software Engineering*, ser. Lecture Notes in Informatics, vol. 106. Bttinger, Stefan and Theuvsen, Ludwig and Rank, Susanne and Morgenstern, Marlies, 2007, pp. 45–54.