Towards Model-Based Testing of Domain-Specific Modelling Languages

Janne Merilinna
VTT Technical Research Centre of Finland
P.O. Box 1000, 02044 Espoo, Finland
janne.merilinna@vtt.fi

Olli-Pekka Puolitaival
VTT Technical Research Centre of Finland
P.O. Box 1100, 90571 Oulu, Finland
olli-pekka.puolitaival@vtt.fi

Juha Pärssinen
VTT Technical Research Centre of Finland
P.O. Box 1000, 02044 Espoo, Finland
juha.parssinen@vtt.fi

Abstract

Domain-Specific Modelling (DSM) has evidently increased the productivity and the quality of software development. The witnessed gains are primarily caused by the three corner stones of DSM, i.e. Domain-Specific Modelling Languages (DSML), code generators and software frameworks. Although the DSMLs and the code generators are the primary reason for the gains, little attention have been paid in making sure that these work correctly. In this paper, we present a work in progress on the technique of utilizing the Model-Based-Testing (MBT) as a means for testing the elements of the DSM basic architecture. We will discuss how the MBT can be utilized for generating a comprehensive test suite of application models, in addition to how the generated applications can be tested with the MBT. As a combination, the DSM basic architecture will be tested thoroughly. We also present how the introduced technique can be realized by utilizing the tools currently available for the DSM and the MBT.

Categories and Subject Descriptors D.2.5 [**Testing and Debugging**]: Testing tools

General Terms Languages, Verification.

Keywords Metamodel testing

1. Introduction

Domain-Specific Modelling (DSM) is all about raising the level of abstraction from thinking of the software in a solution-space into a level concerning the software in a problem-space. Although, during research, there are no or only a few tiny experiments [1] in comparing the benefits of DSM and traditional software development means, cases conducted in industrial settings constantly show 5-10 times productivity gains compared to the traditional software development [2,3]. In addition to the productivity gains, the DSM is expected to have a positive impact on the software quality [3].

Productivity and quality gains are primarily caused by the three corner stones of DSM, i.e. Domain-Specific Modelling Languages (DSML), code generators and software frameworks which when combined are known as the DSM basic architecture [3]. Instead of modelling the software in solution-space, DSMLs provide concepts that directly map the concepts found in the problem domain. These concepts also include, in addition to elements found in the domain, restrictions that guide the application developer in developing applications that function correctly. The responsibility of the code generator is then to transform these high-level specifications into a source code running on top of the target platform. By automating the model to source code transformation, a great deal of error-prone translations that otherwise had to be implemented manually can be omitted.

Testing in a traditional software development is one of the corner stones in order to raise quality. Regardless of the fact that it is the DSML and its code generator that are the primary sources of errors, there are not that many publications that consider testing metamodels and code generators. In [4], Sadilek and Weißleder present a technique for testing metamodels. These models are comprised of positive and negative test models that are utilized for evaluating the correctness of the metamodel. The positive test models are comprised of models that should be possible to be modelled by using the metamodel, where the negative test models are something that shouldn't be possible to model. Stuermer et al. in [5] present a systematic code generator testing technique, which is applicable in situations where the source and target languages are executable. This essentially means the comparison of the behaviour of a model that can be simulated for an executable run in a target environment. In [3], Kelly and Tolvanen consider testing DSMLs and code generators as a combination. They suggest that the language creation should be considered incremental and test case driven. This means the development of the language and the code generator in small incremental steps and testing these by developing some small applications or by rebuilding applications that have already been developed. In this way, it is expected that the DSM basic architecture will be tested thoroughly.

In situations where one can not afford to release the language and its supporting code generator prior to making sure that they function correctly, not enough confidence can be attained without systematic testing. Particularly, in the case where software development is iterative and incremental, a question whether the languages and code generators under continuous change still function correctly in all cases is raised. In addition, there is an issue of test suite maintenance when metamodels, code generators and application models evolve.

Model-based testing (MBT) [6] is a prominent black box software testing method that enables the creation of a comprehensive test suite by modelling the behaviour of a system, where the models can then be transformed into a test suite by utilizing several test design algorithms [7]. This enables automating the generation of a test suite from models that are easier to keep in synch with the evolving software system. As the test suites are based on models, the maintenance effort of the test suite also decreases.

In this paper, we present a work in progress on applying the MBT for testing the DSM basic architecture in the context of an

iterative and incremental software development process. We also discuss the topic from the view point of how the testing technique can be realized by utilizing the existing tools for the MBT and the DSM.

This paper is structured as follows. First, the basic principles of MBT are briefly discussed. Second, the technique for testing the DSM basic architecture is presented. Third, how the technique can be realized in practice is presented, followed by a discussion and conclusions. The final remarks close the paper.

2. Model-Based Testing

In the past, regression tests for test automation systems have been developed manually [8]. When implemented manually, there is always an extra effort in maintaining the test scripts. So-called keyword-driven testing [9] has been seen as a prominent method for solving the test script maintenance issue. However, keyword-driven testing still requires manual implementation, therefore the problem is only partially solved. MBT is seen as a complementary approach for solving the test script maintenance issue by automating designing and implementation of the test suite [6].

The MBT is a black box software testing method in which the test scripts are automatically generated from a model which describes the behaviour of the system under test (SUT) [10]. The test scripts are generated from a model by utilizing a set of test design algorithms [7] that traverse the model and generate test scripts from that basis.

The MBT process can be divided into three phases, i.e. modelling, test generation and test execution (Figure 1). The modelling stands for the modelling of system behaviour, where the test generator then generates a test suite from these models. The test executor then conducts the test. Next in this section, the phases of the MBT are discussed briefly.

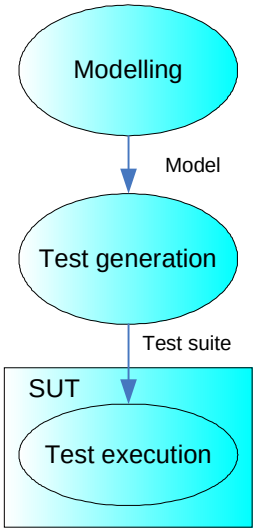

Figure 1. Model-based testing process.

2.1 Modelling

The functional requirements of the software systems is the primary source for developing MBT models [6 p.27]. These models embody the externally visible behaviour of the system. As the system requirements are also modelled with the MBT, in addition to realizing the requirements as an implementation, exist two opinions of the behaviour of the system. The differences between these can be viewed as errors.

The model is required to have knowledge of the input and output data of the SUT. The input data is used for executing the tests and the output data is for validating the tests. The model can be made from an environmental [11] or design [12] viewpoint. The viewpoints are mirror images and are equally suitable for test generation. The design viewpoint for the MBT is similar to the modelling viewpoint for implementation purposes, but at a higher abstraction level. The implementation model can therefore be reused to model design viewpoint MBT model [6]. The notation of the models can be graphical, textual or mixed, where the notation varies from general purpose to domain-specific [10].

2.2 Test Generation

Test generation is based on a model traversal, where several algorithms, called test design algorithms, are utilized for generating test cases from the model. There are three main categories of test design algorithms [13]:

- Requirement-based criteria i.e. the test generator strives to cover all the marked requirements,
- Coverage criteria, i.e. a test suite is generated on the basis of covering a certain degree of the model, and
- Walking algorithms, i.e. an algorithm determines how the model is traversed and the test suite is generated from that basis.

2.3 Test Execution

A test execution can be performed either offline or online [10]. The offline testing stands for generating tests first and then executing the tests separately, whereas with online testing, one step at a time is generated based on the output of the SUT. The difference between the test approaches is that in the offline approach, test generation is separated from test execution and therefore there is a possibility to utilize algorithms requiring heavy computation without the test execution suffering. In the case of online testing, the following test step is generated on the basis of the previous step output values of the SUT, therefore in order to have an immediate response to outputs, only algorithms requiring less computation can be utilized. Online testing also enables the ability to have infinite test suites and enables handling the non-deterministic behaviour [14] of the SUT.

3. Technique for Testing the DSM Basic Architecture by Utilizing Model-Based Testing

Software testing, in the case of DSM, essentially means testing the primary sources of errors, i.e. testing the metamodel and the code generator. Although being the primary sources of errors, not much research have taken place to address these [3, 4, 5].

In the context of iterative and incremental software development processes, there is also concern about the test maintenance because the applications evolve. Additionally, code generators are under constant evolution, when the underlying platform evolves. Changes in the metamodel will also have an impact on the code generators, whereas changes in the metamodel and the code generators have an impact on the existing applications. When one of the three aspects evolve, one has to therefore have tests in order to make sure that the evolved versions function correctly.

In this paper, it is argued that testing application models, metamodels and code generators cannot be performed separately since all of the three levels are intertwined tightly together. This is because modelling languages consist of syntax and semantics definitions. The metamodel describes the semantics of the model, but it cannot have an impact on how the code generator decides to produce the code. It is therefore the code generator that makes the ultimate decision for what is generated and how and thus the metamodel and the code generator, as a combination, define the semantics in practice. This speaks on behalf of the testing practice introduced in [3]. However, the presented approach does not take into account that the code generator and the metamodel cannot be tested thoroughly by only a couple of example applications.

MBT can be seen as a prominent method for generating comprehensive test suites. This statement is based on the capability of the test design algorithms to produce a comprehensive test suite from the MBT models. A model being easier to maintain compared to a test suite, decreases the maintenance effort [6]. In addition to lightening maintenance, the test suite will always be in synch with the MBT model. Next in this section, techniques for utilizing the MBT in testing the DSM basic architecture is discussed.

3.1 Generating a Test Case for the DSM Basic Architecture

The MBT can be utilized for testing the applications developed with the DSM approach. However, testing the applications differs in the case of DSM, from the traditional MBT. In the traditional MBT, models and the implementation are derived from informal software specifications, therefore it is tested whether the application implementation follows the specifications. In the DSM approach, the implementation and the MBT are derived from the same model, thus it cannot be tested whether the application is implemented according to the specifications but it is tested whether the code generator produces a working application running on the software framework from the application model. Thus, in the context of DSM, one application model can be seen as a test case for the whole DSM basic architecture.

The technique for utilizing the MBT in developing a test case for the DSM basic architecture is illustrated in Figure 2. The left side of the figure follows the basic code generation process, whereas the right side follows the MBT process. Considering the MBT process, the difference between the introduction of the DSM to the MBT and the traditional MBT is in the source of the model. Whereas in the traditional MBT the model is derived from software specifications, in this case the model is derived from the same model that the DSM utilizes for the code generation. By doing so, both the code generator and the MBT tool always have the same conception of the model. Errors are detected if the code generator realizes the conception differently, thus the MBT model no longer matches the generated application when executed.

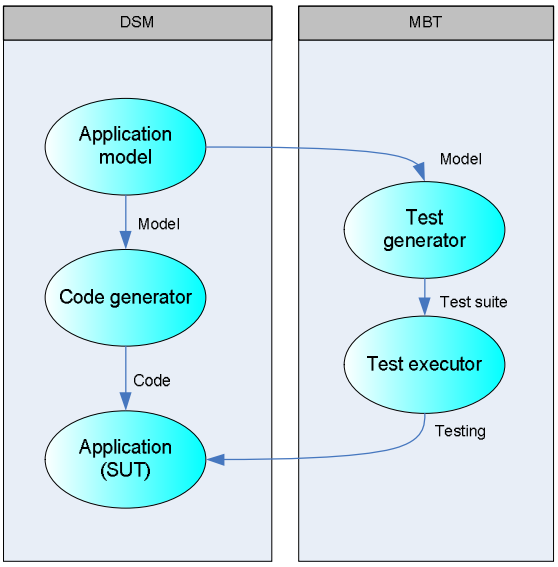

Figure 2. MBT utilized for testing applications.

3.2 Generating a Test Suite for the DSM Basic Architecture

One test case does not test the whole DSM basic architecture thoroughly. By modelling of a set of applications, test coverage increases. However, the effort to maintain the test suite, i.e. a set of application models, can become an issue when the DSM basic architecture evolves.

As in the case of the DSM, all applications of a certain domain are based on a metamodel, the test suite for the DSM basic architecture should also be based on this metamodel. In this paper, it is argued that the MBT can be utilized for generating a comprehensive test suite of application models for the DSM basic architecture from a metamodel. This claim is based on the capability of the MBT to generate a test suite from models, of which metamodels essentially are. If the test suite is generated from a metamodel, it will also always be in synch with the evolving language, thus decreasing the maintenance effort of the test suite.

An overview of the technique for generating a test suite for the DSM basic architecture is depicted in Figure 3. It must be noted that it is the application models that are generated from the metamodel by the MBT tool. As the MBT tool utilizes the metamodel for generating the test suite, the metamodel has to be strictly defined, i.e. the metamodel should be defined in such a way that only legal applications can be modelled. If not, the metamodel test suite also includes test cases, i.e. application models, that are not legal for the target platform. With enough application models, the errors in the DSM basic architecture can be noticed.

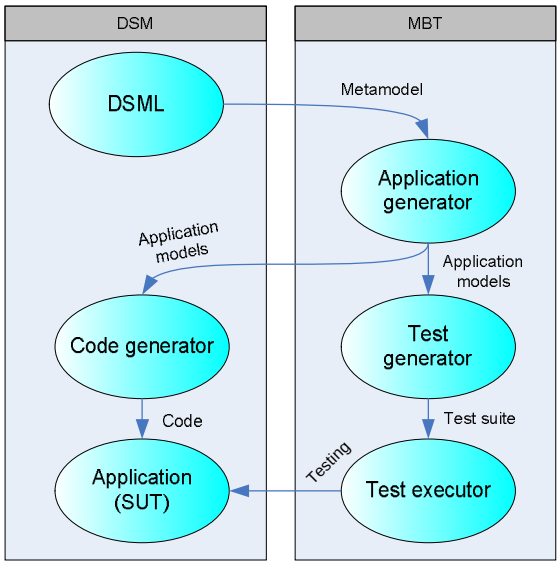

Figure 3. Testing the metamodel.

4. Test Suite Generation for the DSM Basic Architecture – Illustration with the Existing Tool Support

The technique for generating a comprehensive test suite for the DSM basic architecture requires an extensive tool support. The DSM requires a language workbench that provides modelling, metamodelling and code generation facilities, whereas the MBT requires an environment that takes, as an input, a model that is utilized for the test suite generation. Additionally, an extensive collaboration between the tools is also required. The way the tools can collaborate has an impact on how the test suite generation can be realized in practice. Thus, in this section, the test suite generation technique is illustrated from the view point of the currently available tool support. First, currently available tool support for the MBT and the DSM are discussed and two of the most suitable tools have been chosen for this illustration. Second, the technique is discussed from the view point of the selected tools.

4.1 Tools for Domain-Specific Modelling and Model-Based Testing

Enabling the test suite generation requires a DSM tool providing facilities for exporting the metamodel into a format required by the MBT tool. In addition, the DSM tool has to enable importing models generated by the MBT tool, whereas the MBT tool has to enable exportation of the test suite, i.e. a set of application models, to the DSM tool, in addition to providing facilities for generating the test suite for the imported application models.

4.1.1 Tools for the Model-Based Testing

Tool support for the MBT is extensive [6, p.401-403] [15]. However, many of the MBT tools are still immature but there also exists commercial tool vendors providing more mature tools and support when required. MaTeLo from All4Tec[1] is for control oriented MBT. Reactis from Reactive Systems[2] and T-Vec[3] provides tools focused in embedded MBT testing. Test Designer from Smartesting[4] and Conformiq Qtronic from Conformiq[5] are general-purpose solutions for the MBT. Scrutinizing the MBT tool evaluation, presented in [13], reveals the Qtronic to be mature enough and provides open data formats for importing and exporting the models.

The Qtronic expects the input model to be either in

- a similar format as the UML state machine diagram extended with a variant of Java, which is called QML, or
- a textual representation, where the programming language is QML.

Both of the model types represent a model where the input and output pairs are defined. In addition, special requirements can be defined for the inputs and outputs that guide the test design algorithms in generating the test suite. The test suite generator can be implemented by utilizing the provided plug-in interface.

4.1.2 Tools for the Domain-Specific Modelling

The Generic Modeling Environment 6 from Vanderbilt University[6] and Metaedit+ 4.5 from Metacase[7] are probably the most well-known language workbenches. Microsoft also provides a DSM tool with Microsoft Visual Studio 2005 SDK[8]. There are also open source tools available, such as the Generic Eclipse Modeling System[9].

Metaedit+ is our choice among the tools since, as far as we know, it is the only language workbench providing code generator facilities with a language dedicated only for developing code generators. This enables a rapid development of generators compared to a situation where the code generators are developed by utilizing the APIs of the modelling tools. In addition, Metaedit+ enables importing models in an XML format.

4.2 Test Suite Generation in Practice

In order for Metaedit+ to export its metamodel to QTronic, a metamodel has to be modelled with the included GOPPRR modelling language. Exporting the metamodel requires a Metamodel-to-QML (Met2QML) code generator that takes a metamodel as an input and generates QML out of it. Now, the Qtronic imports the generated metamodel. By utilizing the means of the MBT, the Qtronic traverses the metamodel extensively and stores the traversed paths as lists of visited concepts. The traversed paths then form a test suite of application models. The test suite of application models is then utilized for generating application

[1] http://www.all4tec.net/

[2] http://www.reactive-systems.com/index.msp

[3] http://www.t-vec.com/solutions/products.php

[4] http://www.smartesting.com/

[5] http://www.conformiq.com/qtronic.php

[6] http://www.isis.vanderbilt.edu/projects/gme/

[7] http://www.metacase.com/mwb/

[8] http://msdn2.microsoft.com/fr-fr/vstudio/aa718368(en-us).aspx

[9] http://www.eclipse.org/gmt/gems/

models in a XML format required by the Metaedit+. The test suite of application models is then imported by the Metaedit+, which then generates a source code from the models. As a result, there is a set of applications ready to be executed and tested.

In order to test the generated applications, the application model of each application has to be imported back to the Qtronic. This requires a model-to-QML (Mod2QML) code generator that generates application models into a format required by the Qtronic. The Mod2QML now generates application models in a format required by the Qtronic, which then generates a test suite for each application. The test suites for applications are then forwarded to the test executor, which then conducts the test. An overview of the workflow is depicted in Figure 4.

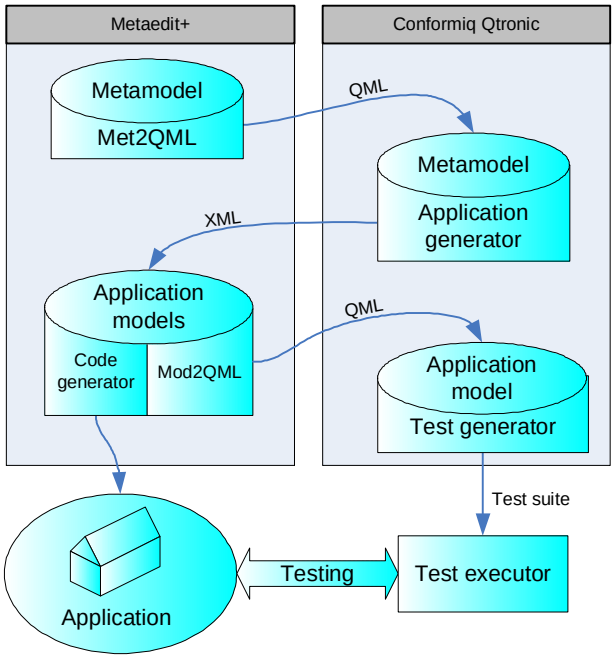

Figure 4. Overview of the workflow for generating a test suit for the DSM basic architecture.

5. Discussion

Test automation has decreased the cost of testing, but there is still an issue of test maintenance. MBT strives to ease the maintenance by test suite generation. A technique for generating a test suite for the DSM basic architecture merely brings another set of entities that have to be tested. This also causes, in addition to more things to test, more aspects to maintain. It is a justifiable question whether this worth for it. Currently, there is no answer to the question. It is not known how much extra effort it takes to have this test and what are the benefits of it compared to rather adhoc DSML testing. In addition to these, a few open questions exist.

- Can all metamodels be transformed into a format required by the MBT tools, i.e. are there any special cases where the metamodel cannot be transformed into a MBT model,
- is it feasible to generate application models from the metamodel by an approach of MBT,
- are there special cases where the MBT cannot be utilized for generating a test suite for the applications from the DSML models, and
- how to automate the whole process.

In order to deal with the open questions, we will continue the development of 1) a generator for generating a test suite from DSML models for applications and 2) a generator for generating a test suite of application models from the metamodel.

6. Conclusion

Industrial cases constantly reveal 5-10 productivity gains when utilizing DSMLs in the software development. Although the gains witnessed so far are great, a concerned question raised is how to make sure the languages, i.e. metamodels and code generators, are correct especially in the cases where the languages are provided for the masses and for mission critical systems. Iterative and incremental software development processes bring an additional question of test suite maintenance. Although testing is an important means to detect errors in the traditional software engineering, it is not well known how to test the DSM basic architecture thoroughly.

Our contribution is a technique for utilizing the MBT for generating a test suite for the DSM basic architecture. The presented technique does not strive for the testing of the layers of the DSM basic architecture in isolation, but to test the whole architecture. The test technique is based on generating a comprehensive test suite of application models by utilizing the means provided by the MBT plus generating test suites for the generated applications. As a combination, the DSM basic architecture will be tested thoroughly. The presented technique is also discussed from the tool support point of view, in order to enable further discussion on the feasibility of the introduced technique.

References

[1] Merilinna, J. and Pärssinen, J., "Comparison between different abstraction level programming: experiment definitions and initial results", The 7th OOPSLA Workshop on Domain-Specific Modeling, Montreal, Canada, 2007

[2] Hammond, J. L., "Domain-specific modeling significantly reduces development time", URL: http://www.metacase.com/papers/ece_april2008.pdf [Visited at 18.6.2008]

[3] Kelly, S. and Tolvanen, J-P, "Domain-Specific Modeling: Enabling full code generation", John Wiley & Sons, ISBN 978-0-0470-03666, 427 p., 2008

[4] Sadilek, D. A. and Weißleder, S., "Testing Metamodels", Fourth European Conference on Model Driven Architecture Fondations and Applications, Berlin, Germany, 2008

[5] Stuermer, I., Conrad, M., Doerr, H., and Pepper, P., "Systematic Testing of Model-Based Code Generators", IEEE Trans. Softw., pp. 622-634, 2007

[6] Utting, M. and Legeard, B., "Practical Model Based Testing: A Tools Approach", Morgan Kaufmann 1st ed., ISBN: 978-0123725011 , 456p., 2006

[7] Utting, M., Pretschner, A. and Legeard, B., "A taxonomy of model-based testing", Working papers series. University of Waikato, Department of Computer Science, Hamilton, New Zealand, University of Waitako, 2006

[8] Wahl, N., "An overview of regression testing", ACM SIGSOFT Software Engineering Notes, Volume 24 Issue 1, ISSN:0163-5948, pp. 69-73, 1999

[9] Mosley, D. and Posey, B., "Just Enough Software Test Automation", Yourdon Press. Prentice-Hall, ISBN:0130084689, 300p., 200

[10] Hartman, A., Katara, M. and Olvovsky, S., "Choosing a test modeling language: A survey", Haifa Verification Conference, pp. 204-218, 2006

[11] Auguston, M., Michael, J. and Shing, M., "Environment Behavior Models for Scenario Generation and Testing Automation", ACM SIGSOFT Software Engineering Notes, Volume 30 Issue 4, Advances in Model-Based Testing (A-MOST 2005), 2005

[12] Schulz, S., Honkola, J. and Huima, A., "Towards Model-Based Testing with Architecture Models", Engineering of Computer Based Systems (ECBS '07), pp. 495-502, 2007

[13] Puolitaival, O.-P., Luo, M. and Kanstren, T., "On the Properties and Selection of Model-Based Testing tool and Technique", 1st Workshop on Model-based Testing in Practice (MoTiP 2008), Berlin, Germany, 2008

[14] Nachmanson, L., Veanes, M., Schulte, W., Tillmann, N. and Grieskamp, W., "Optimal strategies for testing nondeterministic systems", ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 55–64, 2004

[15] Hartman, A., "Model Based Test Generation Tools", Agedis Consortium, URL: http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf, 2002