Integrating Domain Specific Modeling in Model-Based Testing

[1]Satyapal reddy Regenti and [2]Dr. R.J. Rama Sree

[1]Department of Computer Science, JBICT, Tirupati-517 501, A.P., India.
regentisatyareddy@yahoo.com
[2]Department of Computer Science, Rashtriya Sanskrit Vidyapeetha, Tirupati-517 502, A.P., India.
rjramasree@yahoo.com

ABSTRACT - Model-Based Testing is a test automation technique that generates test cases based on a model of the system under test. Domain-specific modeling is a modeling approach where the developed system is modeled in terms of domain-specific concepts and these models are automatically transformed to other forms such as application code. In this paper, we will discuss the adoption and integration of domain-specific modeling with model-based testing tools. Since model-based testing tools utilise various modeling notations that typically diverge from a specific domain-model, we will discuss how domain specific models can be automatically transformed to become suitable models for a chosen model-based testing tool. Furthermore, by doing this in terms of a domain-specific meta-model, we will allow one to switch between various model-based testing tools.

Keywords - domain-specific modeling; model-based testing; meta-model

1. INTRODUCTION

Model-based testing (MBT) is a growing trend in test automation. In MBT, the system under test is modeled at a suitable abstraction level for testing, and tools are used to automatically generate test cases based on this model. Given a set of suitable tools and methods, MBT has been shown to be a useful and effective means for high-level testing in different domains [1,2]. Some advantages include reduced maintenance costs in focusing on a single high-level model, and increased test coverage over the aspects expressed in the test model via the means of automated test generation.

Most of the current MBT tools are general purpose MBT tools, focusing on generic models of software behaviour, such as finite state-machines [3]. However, each MBT tool applies its own specific modeling notation, which prevents the application of test models across different MBT tools. This also presents the problem of choosing a suitable MBT tool based on the different needs, such as suitable modeling notation, price, and other features [4,5]. On the other hand, domain specific modeling (DSM) provides the means for expressing domain concepts in a high-level model and for transforming these models to other formats. The typical usage of DSM is to model the target system and to generate the application code itself from these models. Typically, DSM models are self-made and controlled languages, which make them cost-effective and efficient in a suitable context.

It has proven difficult to re-use MBT approaches and test models over different projects due to their domain-specific properties [4]. The test model is made to express the system under test (SUT), which requires the domain concepts in the model to be expressed. As such, they can only be used for specific purposes and are already linked to the specific needs of the target domain, although expressed in general modeling languages such as state-machines. In this paper, we will describe an approach for integrating and adapting DSM for use with MBT tools and techniques. We will show how test models expressed in the terms of domain-specific modeling can be used to provide the form of a domain-specific meta-model for the various MBT tools, focusing on the domain specific aspects of the target system. From this domain specific model, we will show how suitable models can be generated for different MBT tools, enabling the use of DSM concepts with model-based testing tools. Our approach also permits addressing the need to be able to switch between various MBT tools according to various needs during a project lifecycle.

2. BACKGROUND

2.1 Model-Based Testing

Model-based testing is a testing technique aimed to automate test generation from a model which describes a relevant aspect of SUT behaviour. That is, the model describes the SUT from the viewpoint of what needs to be tested.

MBT contains three main parts: modeling, test generation and test execution [3]. These main parts are presented in Figure 1. The model is the primary source of information for the test generator that generates test cases. To be effective, this model has to contain relevant data, but it also needs to be described at a suitably high abstraction level. These models are defined in various formats by the different MBT tools, and some come with their own modeling tool. The most typical type of modeling language uses some form of state machine extended with a programming language. This state machine typically encodes the program state and the programming language is used to describe the expected output data as well as the possible input data.

MBT tools use different test generation algorithms to generate test cases based on the provided model and the given test generation attributes. Test generation attributes are normal parameters for test execution algorithms, i.e., coverage criteria, test case length and calculation depth.

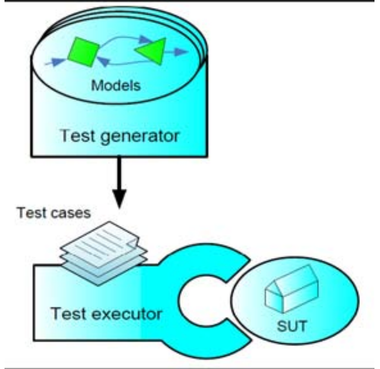

Figure 1. Model-Based Testing overview.

The MBT tools can produce the generated test cases in different formats. The tools typically have a plug-in architecture for writing test script generators, or provide off-the-shelf generators producing an easy-to-parse format for test case export. Flexible test case formatting is valuable, in order to allow one to make use of existing test execution platforms in executing the generated test scripts. Finally, when test scripts have been generated in a suitable format, they can be executed and the test executor will generate a report on the test results.

Several different MBT tools exist, and all of these have some significant differences [5]. This paper is focused on enabling one to change MBT tools during a product lifecycle. Some of the major motivations for why one might want to change a MBT tool include the use of different test generation algorithms for a more comprehensive test generation and model analysis, and an enhanced off-the-shelf support for different output formats. On the other hand, a major constraint in choosing a specific tool can be the offered pricing and licensing. We will identify the three main types of licensing: commercial, open source and self-made.

Commercial MBT tools naturally have the best support available, while they also typically come with more sophisticated interfaces for model importing and editing, and provide support for test case exports. For various reasons, one may also wish to avoid licensing payments, to have the ability to customize the tool, or to be able to more easily share models with different partners. These are examples of cases where an open source solution might be a better choice. There are already several open source MBT tools available that have reached a suitable maturity level to be considered to be useful. While these tools may not be as sophisticated as commercial tools, they are sufficient enough for many cases, especially when beginning to adopt the MBT testing process. The final type which is discussed here is an in-house MBT tool that has been especially developed for specific needs and purposes, such as a specialized problem domain. In such a case, the most natural solution is to build the MBT tool by oneself.

2.2 Domain-Specific Modeling

Domain-specific modeling can be defined as using models to raise the level of abstraction beyond programming, by directly specifying the solution using domain concepts [6]. The actual product that is being developed is then generated from these high-level domain models. The automated generation of the product (code) from the domain model is possible as both the language and generators can be defined to fit the requirements of only one company and domain. The process of applying DSM can be split into two main parts, where the expert first defines the domain specific modeling concepts and implements the DSM modeling tools that enable the developers to use them in modeling and thus also in the building of the actual products themselves.

These two steps can be further decomposed into four main phases:

- metamodeling
- modeling
- code generation
- framework

In the meta modeling phase, the modeling language is created. A metamodel is defined, which specifies a modeling language that can be used to express domain concepts as a basis for code generation. The metamodel defines the DSM language, but can also define restrictions concerning its application, and may also include a graphical view for visualizing the defined models. A well defined language is necessary for effective code generation. The modeling phase is about tranforming the informal application description (e.g. from a natural language specification) into a model in terms of the DSM language defined in the meta modeling phase. As the model is built based on domain concepts, it should – when done well – enable non-developers to create the models. In the code generation phase, the DSM model created in the modeling phase is transformed into another format. Generic code generation from a generic model (such as a state-machine) for any application can be seen as an impractical goal due to the adaptation that it would require. DSM is most effective when the metamodel is made for a limited domain and, therefore, the code generation also needs to be only suitable for that specific domain.

Code generation from a DSM is typically made on a platform. The platform is the more stable part of the code that does not change between the applications. The use of a platform makes the code generation process lighter as it reduces the amount of the code that needs to be generated for each domain application. The cost-effectiveness of DSM is at its best when several applications need to be built for the same domain with some variance, such as a product family.

2.3 Related research

Katara et al [7] have used DSM to make MBT easier to adopt in an industrial context. While they focused on providing a more effective modeling language, in this paper we will focus more on making use of the benefits of DSM in the scope of MBT. This also means providing more effective modeling languages (as domain specific meta-models), whilst also supporting different tools through these DSM test models. We have previously used MBT for automating the testing of DSM application models and metamodels [8,9]. In those studies, we focused on integrating MBT into the DSM development workflow. In this paper, we will aim to bring the DSM benefits into the MBT process, to allow one to evolve their MBT process more easily and effectively.

3. INTEGRATING DOMAIN-SPECIFIC MODELING IN MODEL-BASED TESTING

Our objective in integrating DSM with MBT is to make use of the advantages of DSM in the MBT domain, allowing the use of specific DSM models across different tools, as opposed to having a specific modeling language which is different for each tool. We can see several advantages in this, in providing a more natural means for reasoning about the systems using a language which is tailored to their domain concepts, mitigating the impact of the constraints and enabling one to make better use of the advantages of the various MBT tools. In our approach, a DSM language is used to describe the tested application, based on the domain meta-model. From this domain specific application test model, a test generator is used to generate a suitable model for the MBT tool of choice. Figure 2 shows an overview of our approach. In the following, we will describe this approach in more detail.

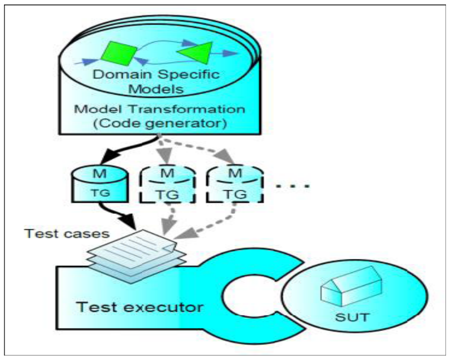

Figure 2. Model-based testing with domain-specific modeling.

In the following, we will describe the application of our approach in the form of two main steps:

- Model the tested application in terms of DSM, to produce a DSM test model.
- Use a specific model transformation from the DSM test model into a suitable model format for the test generator (TG) of a chosen MBT tool.

At first, the modeling language is created following DSML principles [6]. The key issue to take care of here is that the model needs to be able to describe the SUT domain using its concepts. The general idea of DSM is to model the application domain in the terms of its own concepts, at a high abstraction level. Similarly, the models used in MBT are also typically defined at a high abstraction level [3], providing a good fit for the two different models. As this DSM test modeling phase is similar to general DSM modeling, we will not go into the details here. A relevant consideration is the relation of the DSML used to describe the test model vs. the tested implementation. In many cases, it is not optimal to use the same models for testing as are used for implementation.

As mentioned above, in our approach, we would like to generate a suitable test model to be provided as an input for the test generators of different MBT tools. To do this, we need to have a specific code generation phase in the DSM process, where, instead of application code, we generate model code for the MBT tool of choice. The specific advantages provided by this approach are the following:

- It allows for the selection of a used test generator from a single model, while
- making the change of a test generator a light-weight process, and
- enables simultaneous use of different test generators, e.g. open-source, commercial or self-made.

DSM has the potential to hide the complexity of various MBT tools, as it uses domain concepts in modeling. This enables non-programmers to create models for testing, and also enables one to use them for other purposes such as documents in themselves or as a basis for document generation. Here, one relevant consideration is the ability of a versatile DSML to express concepts which are not available in all MBT tools. In our approach, this is mitigated by the support for generating suitable test models for different MBT tools, allowing one to switch to a more advanced tool without losing the investment into existing models. However, these limitations need to be considered in creating the models, model transformers, and in choosing a suitable MBT tool.

4. CASE STUDY

In this section, we will present a case study where we have applied our approach to test generation for a (SIP) application. In this case study, we created a DSM meta-model using the MetaEdit+ tool. Using this meta-model as a basis, we created a DSM test model describing the SIP protocol. From this model, we generated suitable test models for three different MBT tools: Conformiq Qtronic, ModelJUnit, and our own custom-made test generation tool. These tools represent the three types of tools described in section 2.1. Conformiq Qtronic is a commercial MBT tool, ModelJUnit is an open-source MBT tool, and the custom-made tool is our own self-made MBT tool. An overview of the different components is shown in Figure 3. In each case, the final objective was to generate TTCN-3 test scripts from the test model using the MBT tool. TTCN-3 is a widely used test script language for the telecommunication domain.

In the following subsections, we will first present the basics of the SIP protocol and our DSM model for describing it, followed by the description of the various MBT tools used in our case study. Along with the description of each MBT tool, we will also describe our experiences in generating the test model for the tool from our DSM model.

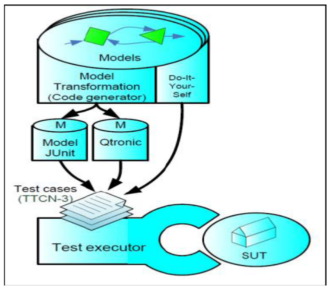

Figure 3. The components of our case study.

4.1 Session Initiation Protocol

SIP is a communication initiation protocol that is used in, for example, IP phone call initiation. SIP is also suitable for various other media connections, such as video calls.

Figure 4 shows a case for initiating a multimedia session between A and B, using the SIP protocol [10]. First, A sends an INVITE message to B. Next, B responds with three messages: 100 Trying, 180 Ringing and 200 OK. A confirms the correctness of the initilization sequence by sending an ACK message and following this, a multimedia session starts. B ends the session by sending a BYE message and A accepts this by sending a 200 OK message.

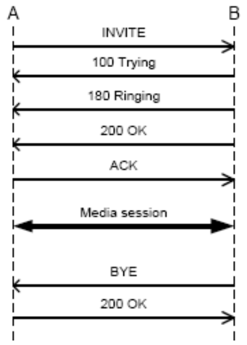

Figure 4. SIP example

Figure 5 shows an example model of the SIP DSM language that we have created using MetaEdit+. The idea of the SIP modeling language is to make SIP message based modeling as easy as possible. Our language is a variation of a state machine, but extended with the domain concepts of SIP, as visible in the model.

States do not have any functionality in this language. All of the functionality is embedded in the state transitions. Transitions have trigger, guard, action, response and requirement fields. A trigger in this model is basically a message from the test executor to the system under testing (SUT). A transition guard defines a condition that must be fulfilled before the transition can take place. An action is a piece of code that is executed in response to a transition being taken. Our DSM language only supports code that performs actions on the variables of the model in a standard way, allowing it to be added directly to the transformed model.

A response defines an expected output in terms of a message that is expected to be received from the SUT to the test executor. The requirement field defines a string value that is tagged to the A part of the model and later used in test cases to define at which point a specific test requirement is covered by the test generator algorithm.

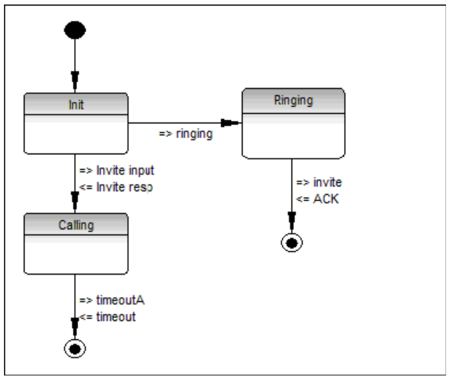

Figure 5. SIP language in MetaEdit$^+$

Trigger and Response (TR) fields are objects which include several different fields of various types. All of the other fields are transformed into single variables in string format. TR fields are objects, as a SIP message has several fields and the test model needs to model the changing fields. Thus, while the test model does not need to model all the fields of the SIP protocol, it does need to model the changing fields. If the set of changing fields evolves, the modeling language can be changed accordingly by creating a new object or changing the fields in the existing objects for trigger and response.

4.2 ModelJUnit

ModelJUnit is a Java based MBT tool. In ModelJUnit, the test model is defined as a Java class. The states and transitions are encoded in the class file, using the Java programming language with specific naming conventions for naming and the Java annotation functions to define the state transition functions as well as the guard statements for when a transition is allowed to be taken. Transition triggers, the test script generator and all other extra features are also programmed in the standard Java notation. This case is illustrated in figure 6.

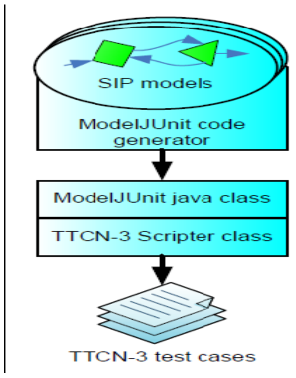

Figure 6. The ModelJUnit generator.

In this case, we created a code generator which transformed DSM models defined in the SIP modeling language to the format of the ModelJUnit Java classes. By executing the ModelJUnit tool with these models as input, we generated TTCN-3 test scripts.

4.3 Conformiq Qtronic

Conformiq Qtronic (CQ) is a commercial MBT tool. CQ uses a UML state machine extended with a variant of the Java programming language as model input language. CQ incorporates its own modeller for state machines, and also supports importing models created in other tools when they are available in a compatible XMI format. CQ has a graphical user interface and is integrated with the Eclipse development environment. It also incorporates various algorithms with tuneable parameters to allow for extensive test generation. CQ has an interface for creating test code generator plug-ins and also includes a couple of off-the-shelf existing plug-ins.

The components of the CQ use case are presented in figure 7. First, we created a CQ code generator to perform the model transformation from our DSM SIP language to the format (XMI+ Java) accepted as input by the CQ tool. We then applied the CQ test generation tool based on this model, and used the off-the-shelf TTCN-3 test generator plugin provided with the CQ to create a set of TTCN-3 test cases from the test model.

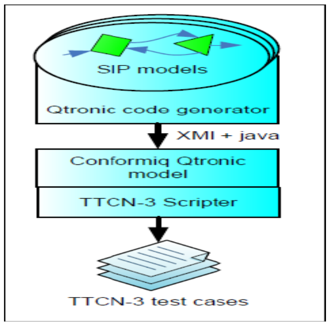

Figure 7. The Conformiq Qtronic generator.

5. DISCUSSION

In this section, we will discuss the preliminary results of our experiments in applying the approach of using DSM as a basis for MBT. Overall, our experience was that the DSM concepts can be applied usefully to produce effective models for MBT. Thus, we believe that significant gains can be achieved with this approach. Based on the three case studies that we performed, it can also be said that in these cases the approach was found to perform well and we were able to generate useful test models for the three different MBT tools. For us, this also highlights the usefulness of the approach based on our previous experiences in applying MBT tools in various contexts.

ISSN : 0975-8283

For example, we have found ModelJUnit to perform well in basic test modeling and test generation. However, it lacks in diversity in the availability of more advanced functions, while it does allow for an extensive customization of the tool itself, where needed, due to its open-source nature. On the other hand, Conformiq Qtronic is a versatile commercial tool with well defined and extensive interfaces and algorithms.

Overall, it can be said that our viewpoint enables one to start experimenting with MBT through the use of DSM. In this way, it is possible to start the experiments with the help of free open-source tools algorithm. After these initial experiments, it is then possible to move to a different test generation approach, such as a commercial MBT tool. By providing the mapping of the DSM model to the various MBT tool models through the different test generators, it is possible to perform this switch while maintaining the investments and lessons learned in the modeling done in the initial phases. Some benefits can be observed in applying the different options and algorithms offered by the various tools. For example, one may make the transition to a commercial tool due to the more powerful and extensive algorithms and the other options available as needs are observed during project lifecycle.

Some cost-effectiveness trade-offs can be observed in having to first create the DSM meta-language for expressing the suitable test models for the target domain. Similarly, creating the test model generators needed for creating the DSM model for MBT model transformation is another factor to be considered. However, these are similar tradeoffs that need to be considered in taking DSM into use in general. Thus the general considerations for the cost-effectiveness of DSM application can be seen as relevant.

To mitigate some of these tradeoffs, some possible solutions can be seen in providing generic parts of a meta-model that can be used for creating suitable DSM languages for MBT. Also, when a test model generator is available for a given domain, it can be used with the different models for that MBT tool, and thus it only needs to be implemented once per domain.

Another point to consider as a potential benefit is that of using the different models and their related transformations as a basis for handling the aspects of traceability between the different points in the software development lifecycle. When explicit transformations are made from DSM models to test models, it is possible to also generate documentation for which parts of the domain model are covered by which test cases and how. As the domain test model will also describe the system in the terms of its domain concepts, it can also be used as a part of the documentation for the system itself. Similar considerations may also apply for the generated MBT models and any extra tools available for these tools to make use of their test models. The addressing of these questions in detail provides interesting research questions for future work.

6. CONCLUSION

In this paper, we have described the concept of using DSM to enable more effective use of different MBT tools as well as the advantages of DSM concepts in connection with MBT techniques. We also described the initial results from our implementation of this concept. For future work, we will look to implement and evaluate the approach on a larger scale.

7. REFERENCES

1. Grieskamp, W., Kicillof, N., Stobie, K., and Braberman, V., “Model-Based Quality Assurance of Protocol Documentation: Tools and Methodology”, Journal of Software Testing, Verification and Reliability,2010.
2. Miller, T. and Strooper, P., “A Case Study in Model-Based Testing of Specifications and Implementations”. Journal of Software Testing, Verification and Reliability,2010.
3. Utting, M. and Legeard, B. 2006. “Practical Model Based Testing: A Tools Approach”, Morgan Kaufmann 1st ed., ISBN: 978-0123725011, 456p.
4. Dias-Neto, A. and Travassos, G., “Model-based testing approaches selection for software projects:, Information and Software Technology, pp.1487-1504,2009.
5. Puolitaival, O.P., Luo, M., Kanstren T., “On the Properties and Selection of Model-Based Testing tool and Technique”, 1st Workshop on Model-based Testing in Practice, MoTiP 2008, June, 2008 – Berlin, Germany.
6. Kelly, S., Tolvanen, J.P., “Domain-Specific Modeling: enabling full code generation”, Wiley, 2008.
7. Katara, M., Kervinen, A., Maunumaa, M., Pääkkönen, T. and Satama, M., “Towards Deploying Model-Based Testing with a Domain-Specific Modeling Approach,” IEEE Computer Society, Windsor, UK, 2006.
8. Merilinna, J., Puolitaival, O.P., Pärssinen, J., “Towards Model-Based Testing of Domain-Specific Modeling Languages”, 8th OOPSLA Workshop on Domain-Specific Modeling. Nashville, USA, pp.19 - 20 Oct. 2008. Tennessee, USA, 2008.
9. Merilinna, J., Puolitaival, O.P., “Using model-based testing for testing application models in the context of domain-specific modeling”, The 9th OOPSLA Workshop on Domain-Specific Modeling. Orlando, FL, USA, pp.25-26, 2009.
10. RFC 3261 - SIP: “Session Initiation Protocol,” 2010, URL: http://tools.ietf.org/html/rfc3261 [Visited at 10.8.2010]