Testing model transformations which derive executable test cases from abstract ones

Erika R. C. de Almeida
Institute of Computing
State University of Campinas
Campinas, SP, Brazil
Email: erikarca@gmail.com

Eliane Martins
Institute of Computing
State University of Campinas
Campinas, SP, Brazil
Email: eliane@ic.unicamp.br

***Abstract*—Model transformations are the core mechanism of model-driven testing. They are repeated many times so that any error can cause large impacts. Therefore, they must be tested before put in use. However, common testing techniques do not apply, once the inputs are test models. Here we present how we are applying a methodology that aims model transformation testing in our use of model-driven testing to obtain executable test cases from abstract ones.**

I. Introduction

Software project managers and developers building applications face the challenge of doing so within an ever-shrinking schedule and with minimal resources. In their attempt to do more with less, organizations want to test software adequately, but as quickly and thoroughly as possible. To accomplish this goal, organizations are turning to automated testing [1].

A convenient definition of automated testing might read as follows: "The management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated tool". The automation of test activities provides its greatest value in instances where test scripts are repeated, once automated testing requires a much higher initial investment than manual test execution does [2].

In an attempt to intend for the benefits of test automation and at the same time reducing its initial cost, we propose a methodology called MOST-WEB whose initial focus is web applications. It combines model-based testing (MBT) and model-driven testing (MDT). We take as basis the software state model, designed according the software's specification, and we get its corresponding abstract test cases [1]. Such abstract test cases are transformed into executable ones, according Selenium[2] format, following the MDT methodology.

The last transformation is not a simple process because the abstract test cases are on the same level of abstraction of the model, while the executable ones should take into account software's implementation details.

This shows MDT is directly dependent on the model transformations, which are the core mechanism for this automation. Writing complex model transformations is error-prone, and efficient testing techniques are required as for any complex program development and is an important challenge if MDT is to succeed [3]. The need for reliable model transformations is even more critical when they are to be reused. Indeed, a single faulty transformation can make a whole model-driven testing process vulnerable.

To test the correctness of a model transformation, there are methodologies that differ greatly, ranging from a formal proof [4] to the application of common testing techniques, taking into account that now the inputs are models and their characteristics [3], such as entities, their attributes and relationships. Here we present the application of one of those methodologies, which was proposed by Fleurey *et al.* [3]. It is an adaptation from a classical testing technique called category-partition testing and was chosen due to its independence of any specific model transformation language and the existence of an associated tool.

The remainder of this paper is organized as follows: Section 2 summarizes our goal of testing model transformations and how it is being achieved using Fleurey *et al.* methodology, and Section 3 presents our conclusions and future work.

II. Model transformation testing

The main goal of our work is the automatic generation of executable test cases from abstract ones. To achieve it we use model-driven testing (MDT), a methodology that can be summarized as follows: the input artifact is a metamodel[3] that specifies how the platform independent test model (PIT) is; PIT is transformed into another test model, also specified by a metamodel, but now platform specific (PST); finally, it is performed a last transformation to get the test code from PST.

Using MDT, we relied on Javed's work [5] and specialized it to web applications. Javed establishes two metamodels, SMC as PIT and xUnit as PST, which respectively represent a sequence of method calls (an abstract test case) and a unit test independent of programming language. Moreover, he also features the model transformations performed by QVT (SMC-xUnit) and MOFScript (xUnit-test code).

[1] Abstract test cases are those that are on the same level of abstraction of the software model, not containing details of its implementation.

[2] Selenium is a suite of tools to automate web applications testing across many platforms.

[3] Metamodel is the model's model that serves for explanation and definition of relationships among the various components of the applied model itself.

In our work we use the same metamodels, especially to our executable test case, a Selenium script, fits the xUnit metamodel. But we made changes in the model transformations: (i) the transformation SMC(PIT)-xUnit(PST), originally implemented in QVT, was rewritten in ATL, which, although not being the OMG (Object Management Group) standard, provides more extensive documentation and Eclipse IDE support; and (ii) the transformation xUnit(PST)-test code, rendered in MOFScript language, was just adapted for Selenium library.

So performing a model transformation, taking models as input and producing other models as output, requires a clear understanding of the abstract syntax and the semantics of both the source and target models, and is an error-prone activity.

Therefore, to be confident that our model transformations are correct, test activity must be performed over them. For this purpose, it was chosen the methodology proposed by Fleurey *et al.* which is based on category-partition testing. Such strategy divide the input domain into ranges and then select test data from each of these ranges. The ranges for an input domain define a partition of the input domain and thus should not overlap. Partition testing has been adapted to test UML models, and Fleurey *et al.* adapted it to test model transformations. In this specific case, the input domain is modeled by the input metamodel of the transformation. The idea is to define partitions for each property of this metamodel.

To represent combinations of partition ranges, they introduce the notions of model fragments, object fragments and property constraints. A model fragment is composed of a set of object fragments. An object fragment is composed of a set of property constraints which specify the ranges from which the values of the properties of the object should be taken from. They are defined in order to check that the set of test models covers the input metamodel of a transformation [3].

Based on these concepts, it is possible to define an iterative engineering process for selecting a set of input models intended to test a model transformation. This process takes two inputs: the input metamodel of the transformation under test and a set of test models. From the input metamodel, it is generated a set of model fragments according to a test criterion. Then the next step checks that there is at least one test model that covers each model fragment.

To automate this engineering process, the authors implemented a tool called Metamodel Coverage Checker (MMCC). MMCC is available as an Eclipse IDE project that has two basic executable artifacts: the first one generates automatically the model fragments of the desired metamodel, and the second one scores the test models coverage.

As MOST-WEB is composed of two independent model transformations (SMC-xUnit e xUnit-test code), this test process must be performed twice. Here it is clear the advantage of the Fleurey *et al.* process being language independent, once our model transformations are written in different languages, ATL and MOFScript.

Table I summarizes the results for the first step. It shows how many model fragments were generated for each metamodel and for each available test criterion, which are all ranges and all partitions. These fragments are our test requirements and we must prepare test cases to cover them. The test cases will be test models, instances of their respective metamodels.

TABLE I
MODEL FRAGMENTS PER METAMODEL ACCORDING A TEST CRITERION.

Test criterion	SMC	xUnit
All Partitions	23	33
All Ranges	43	73

Now we are preparing these test models to score their coverage on the respective metamodel. With a robust test set we will finally be able to assert if the transformation output is correct. This last step, which represents a test oracle, is not performed by MMCC tool and will be done manually.

III. CONCLUSION AND FUTURE WORK

This paper presented how we are testing model transformations, which are used to derive executable test cases from abstract ones in a use of model-driven testing (MDT). A model transformation takes as input a model conforming to a given metamodel and produces as output another model which also conforms to a metamodel. Moreover, it can be considered nothing else than a regular program which requires tests to assure it is reliable. However, using common testing techniques is not so easy due to the fact its inputs are models. For this reason, we choose an approach that aims at testing model transformations.

The testing process is basically composed of two stages which generate model fragments of the desired metamodel and scores their coverage by test models. If the coverage is low, the test analyst must design more test models. Otherwise, it is possible to assert if the target models are being correctly generated by the model transformations.

We are now in phase of model fragments generation. Then we are going to design a set of test models to obtain their coverage on the respective metamodel. At last, with a robust test set, we will be able to see if the model transformation is reliable.

ACKNOWLEDGMENT

The authors would like to thank RobustWeb project and CAPES for supporting this study.

REFERENCES

[1] E. Dustin, J. Rashkaand, and J. Paul, *Automated software testing: introduction, management and performance*. Addison-Wesley Professional, 1999.

[2] R. Ramler and K. Wolfmaier, "Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost," in *AST '06 Proceedings of the 2006 International Workshop on Automation of Software Test*, 2006, pp. 85–91.

[3] F. Fleurey, B. Baudry, P. Muller, and Y. Traon, "Qualifying input test data for model transformations," *Software and Systems Modeling*, vol. 8, no. 2, pp. 185–203, 2009.

[4] M. Hulsbusch, B. Konig, A. Rensink, M. Semenyak, C. Soltenborn, and H. Wehrheim, "Full semantics preservation in model transformation a comparison of proof techniques," University of Twente, The Netherlands, Tech. Rep. 70058, 2010.

[5] A. Javed, P. Strooper, and G. Watson, "Automated generation of test cases using model-driven architecture," in *AST '07 Proceedings of the Second International Workshop on Automation of Software Test*, 2007, pp. 3–9.