Testing Model Transformation: Readings Synthesis

Amr Al Mallah

School of computer science, McGill University, Montreal Canada.
http://www.mcgill.ca

Abstract. The following synthesis evaluates the challenges that are currently present in the area of model transformations. it starts by describing model transformation in the relevant context, and emphasizing of the importance of testing these transformations in MDE. the work then describe a proposed framework in the literature and its limitations.

Key words: Model transformation, Testing model to model transformations, testing framework

1 Introduction : What is model transformation?

MDA (model driven architecture) proposes a new approach, for building software systems. It attempts to move away from developers interpretations of the problem domain. Instead of mapping it directly into code,it encourages creating models, which are closer to the problem domain as the main building blocks of the software system. It really aims at considering models and transformations as first class artifacts for achieving this target.

MDA preaches reliability, reusability and maintainability of complex systems. It aims at realizing that by using automation, of the most error prone parts of the software development process. The focus is to represent the system as a series of models (instances of meta models), and to use model transformations to define, manage automatically the relationships between them, and provide the technology mappings, such as code generators [2].

A Model transformation is, an operation that takes a set of models as input, execute a set of rules over the models elements and produces a set of models as output. The transformation is represented by a set of rules, which could be applied in many ways. They could be applied in order or not, they can even be event driven. Some control flow could be used to define more structured and expressive transformations. Such as in MOTIF [3], Where coupled DEVS are used to describe the control structure of applying the rules in the transformation. In fact running programs are a bunch of encoded transformations that reflect the state of the program, and get applied as applicable throughout the life time of the program.

Its important to distinguish two kinds of transformations, transformations that specify the semantics of a Meta model. These transformations rules usually

add live to the graphs by describing how they can evolve (ex. the Pac Man game simulation). This type of transformation is often called graph rewriting or graph grammar. While another type of transformations describe transforming models of a certain Meta model into another Meta model (example from traffic systems into Petri nets) one can observe in such scenario this is a different kind of transformation.

2 How Important is Model Transformation Testing?

In general, model transformation techniques can be categorized as either model-to-model transformation or model-to-code translation. Model-to-model transformation translates between source and target models, which can be instances of the same or different meta-models. In a typical model-to-model transformation framework, transformation rules and application strategies are written in a special language, called the transformation specification, which can be either graphical or textual.[1]

The transformation specifications are written by humans and are susceptible to errors, especially when the transformation deals with more complex transformations and models. There is a difference between the model transformation specification and the model transformation implementation. The specification of these transformations can be realized by different means, a lot of transformation languages have been proposed [add a reference and/or the table that summarize them] each have certain approach of defining the transformations and the restrictions over them.

3 Model Transformation Testing: Issues And Challenges

Oracle function definition: after we execute a test case it is essential to check the correctness of the results [2]. A test oracle determines whether a system behaves correctly for each test case execution, the oracle function contains a set of input and output pairs.

Automatic Execution of the test case: its absolutely important to have a mechanism that will allow for automatic execution of the input models of the test case pairs in the oracle. And then collect the resulting model and associate it to the test case.

Automatic comparison of models: the procedure of comparing two models for equivalence can be tedious and error prone for a human to perform. Its suggested that this step can be automated using an efficient and applicable model comparison algorithm [1]. One last catch that is related to this comparison is detecting models, which are syntactically different but semantically equivalent.

Visualizing the model differences: in [1] the authors propose describing models as graphs with vertices and edges, and gives an algorithm to detect the differences between two models that are represented as such. And then visualize the differences within a modeling environment.

The automatic generation of test data: which test cases to use, or to generate which could be useful in testing a specification could make the difference. And its pretty cumbersome task to produce these test cases for complex transformations. Since they tend to be incomplete in most cases.

4 A proposed framework for testing model transformations

4.1 The framework

In [1], the authors have described a framework for Model transformation testing that focuses on transformation specification testing within the context of model-to-model transformation where source models and target models belong to the same meta-model.

This testing engine is realized in the context of the Generic Modeling Environment (GME), which provides meta-modeling capabilities. The actual transformation engine is called C-SAW, which is presented as an integrated plug-in. within this context, the transformation specifications are described using a form of OCL.

They suggested that their approach should not be limited to these tools, but should work with any combinations of modeling tool and transformation engine, given that there is a mechanism for integration (plug-in architecture).

Their Framework for model transformation testing has three main components:

Test case constructor The framework assumes that the tests are planned by the user and defined in a textual test specification. Where a simple test case specification defines a transformation specification to be tested, a source model and the expected model. As well as the criteria for determining that the test case passed successfully.

Test engine Will load and execute the test case dynamically. The comparator collects the result and compares it to the expected result and the given criteria. During the execution and comparison step the meta model is used to provide the constraints for test case execution and model comparison.

Test analyzer The analyzers job, is essentially analyzing the results produced by the comparator, and allows for better localization of the differences. Its used, in the proposed framework, to visualize these differences in the modeling environment itself.

4.2 The technicalities

The framework is pretty much based on the concept of model mapping and difference, using the following techniques:

Graph representations of models Models, in GME, could be represented as typed and attributed multi-graphs. Consisting of a set of vertices and edges.

Vertex: is a 3-tuple (name, type, attributes) , where name is the identifier of the vertex, type is the name of the meta-model element, and attributes is a set of attributes that are predefined in the meta-model element

Edge: is a 4-tuple (name, type, src, dest) same as above, with src as the source vertex and dest as the destination vertex.

Model mapping and differences When we compare two models, two sets get produced: the mapping set MS, and the difference set DS . A pair of mapping is denoted as (elem1, elem2), where elem1 is in M1, and elem2 is in M2. They can be a pair of edges or a pair of vertices. Where they have the same name and types.

Model comparison In GME, for model comparison algorithms, a pragmatic approach exists to solve this problem, which is representing models as XML files, which allows for comparing two models textually rather than using an expensive graph-matching algorithm. Issue here is again how to compare two models that look different, contain different elements but have the same meaning. The work presented in [1] doesnt propose a solution to this.

The work presented an algorithm to calculate the mapping set and the difference set of two models. The algorithm uses for its matching criteria, a universally unique identifier that is created and assigned to newly created elements. Which will make the matching by name/id /type .

5 A proposed framework for automatic generation of testing data

A complementary question falls naturally here, after the preceding explanation, is who defines this test oracle. By test oracle I mean the (input, output) test cases. When dealing with a complex transformation it becomes more complicated to generate these test cases. Thats why in [2], the authors look at the problem generating these inputs, automatically such that a maximum coverage is achieved.

Their approach involves analyzing the meta-model of the input model, and generates, using an algorithm, test models as inputs for test cases for this transformation. The paper describes black and white box testing techniques, in order to analyze and partition the domain of the possible input values. The main contribution is through the bacteriological approach to enhance the set of inputs, and even generate more optimized set of inputs for test cases for the specified transformation.

6 Conclusion

Its clearly evident that by solving some of the problems in the framework proposed in [1], specifically model comparison of semantically equivalent models. And allowing transformations across different meta models. This framework could be combined with a framework for automatically generating test cases like descried in [2]. This combination could lead to a more serious and complete framework for testing model transformations.

References

1. Yuehua Lin, Jing Zhang, and Jeff Gray. A Framework for Testing Model Transformations, in Model-driven Software Development, (Sami Beydeda, Matthias Book, and Volker Gruhn, eds.), Springer, ISBN: 3-540-25613-X, 2005, Chapter 10, pp. 219-236, 2005.
2. Benoit Baudry, et al. Model transformation testing challenges. In ECMDA workshop on Integration of Model Driven Development and Model Driven Testing, Bilbao, Spain, July 2006
3. MoTiF : Eugene Syriani and Hans Vangheluwe: Programmed Graph Rewriting with DEVS. In: Manfred Nagl and Andy Schrr (eds), 4th International Conference Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007). Lecture Notes in Computer Science (LNCS). Springer-Verlag, (2008). Kassel, Germany.