Test Case Generation by Grammar-based Fuzzing for Model-driven Engineering*

Magdalena Widl

Knowledge-based Systems Group, Vienna University of Technology, Austria
widl@kr.tuwien.ac.at

1 Introduction

Software models, traditionally used mainly for documentation and informal specification purposes, are becoming first-class development artifacts in the area of Model-driven Engineering (MDE). In MDE, code is generated automatically from multi-view models described in languages like the Unified Modeling Language (UML)[1]. Maintaining consistency between the different views of a model is crucial for the generation of correct code. As software models undergo evolution, particularly in cooperative development environments, tool support for evolution tasks like versioning and merging is indispensable. It is important to thoroughly test such tools in order to avoid the introduction of inconsistent models. However, real-life test cases that cover sufficient evolution scenarios are difficult to obtain. We therefore suggest a method to generate artificial scenarios to facilitate fuzz testing of model evolution tools. In previous work [2] we presented an approach to merge concurrently evolved sequence diagrams with respect to the behavior modeled in their corresponding state machines view. We described the *sequence diagram merging* (SDM) problem formally, suggested a method to solve this problem, and implemented a prototype based on the EMF framework[2]. As there were no benchmarks available, we manually created a set of test cases. However, this is a very cumbersome testing method particularly when a good coverage is needed. A set of randomly generated instances solves this problem, as we show in the following.

2 Grammar-based Fuzzing of Model Evolution Scenarios

Fuzz testing is a black-box software testing technique based on large amounts of randomized input data and has been successfully applied in many areas, e.g. in error detection for UNIX applications [1]. We propose to create randomly generated sequence diagrams and state machines based on a language definition given as metamodel and on a formal specification of the dependencies between the two views. Sequence diagrams model possible communication scenarios between different instances of state machines. A sequence diagram is *correct* if the messages are totally ordered and the sequence of

* This work was supported by by the Vienna Science and Technology Fund (WWTF) through project ICT10-018

[1] http://www.omg.org/spec/UML/

[2] http://www.eclipse.org/modeling/emf/

received message symbols on each lifeline occurs as path of triggers in the state machine that models its behavior (cf. definitions given in [2]). Based on the Ecore implementation of the multi-view metamodel presented in [2], we first create a state machine view as a set of state machines. Using this view, we create a correct sequence diagram that instantiates the state machines with its lifelines. The number of state machines, upper and lower bounds for the number of both states and transitions, the number of messages, and the number of lifelines are defined as input parameters. The generation of states, transitions, transition labels and the assignment of state machines to lifelines is done at random. When generating the message sequence of a sequence diagram, the following restriction is required to ensure its correctness: the symbol of each message must continue a path of triggers in the state machine modeling the behavior of the lifeline that receives the message. Hence only symbols can be considered that occur on outgoing transitions of states to which the previous message symbol has led. Out of these symbols, one is chosen randomly. We can also generate evolutionary changes to the models: a versioning scenario of a sequence diagram is created by copying the diagram and adding messages.

Using this approach, we generated 100 instances of the SDM problem with different parameter settings to find errors in and to assess the scalability of our approach presented in [2]. These tests helped us to find some minor implementation errors and one error in the algorithm solving the SDM problem. With the random instances being much larger than the manual ones, we could also easily identify a performance bottleneck in the first version of the implementation. The errors were not found by using the hand-crafted instances because their detection required a certain combination of lifelines and message sequences that does not occur often and is not naturally thought of by a human. The detected errors resulted in merged sequence diagrams that were inconsistent with the set of state machines.

3 Conclusion and Future Work

We proposed to use a grammar-based fuzzing approach for testing and evaluating MDE tools. This has shown to be very effective in detecting errors in our implementation of an algorithm for the SDM problem. While being specific to a metamodel in our implementation, our approach can be easily adapted to a different Ecore metamodel and thus serve to test different MDE tools. Depending on particular test cases some adaptations may be useful. For instance, to test the SDM problem, instances that actually have a solution (a correctly merged sequence diagram) are desirable. This could be done by first generating a sequence diagram that represents the solution, and then infer an instance (an original and two modified diagrams) from it.

References

1. B. Miller, D. Koski, C. Pheow, L. V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisited: A re-examination of the reliability of unix utilities and services. Technical Report CS-TR-1995-1268, University of Wisconsin, 1995.
2. M. Widl, A. Biere, P. Brosch, U. Egly, M. Heule, G. Kappel, M. Seidl, and H. Tompits. Guided merging of sequence diagrams. In *Pre-proceedings of SLE 2012*, pages 163–182, 2012. Available at `http://modelevolution.org/publications/sle12.pdf`.