AUTOMATED ROBUSTNESS TEST GENERATION USING OCL CONSTRAINTS

János OLÁH
Advisor: István MAJZIK

I. Introduction

The use of different engineering models in software development is prevalent. As models are getting more detailed and sophisticated, they describe the system more precisely. Creating these models takes more effort from software developers, hence the demand to reuse these detailed models in every step of a software development process is a common goal.

Testing is a primary software verification technique used by developers today, but unfortunately a formal comprehensive method does not exist. Model-based testing is a state-of-the-art way to address the problem of software testing (a good overview is presented in [3]). It appeared first by using precise mathematical models (e.g. FSM, LTS) to determine optimal test-cases (optimal in a sense of best coverage in unit time). Several solutions exist how to transform other engineering models into these precise formal models in order to allow automated test generation.

Other potential possibilities emerged with the introduction of the design by contract (DBC) approach (e.g. JML [4]). The principal idea behind DBC in object oriented systems is the following. A class and its clients have a contract with each other. The clients must guarantee certain conditions, before calling a method (preconditions), and the class must ensure certain conditions that will hold after the method call (postconditions). In DBC, these contracts are mapped to executable code, thus any runtime violation can be detected immediately. In case of JML, an automated test generation based on JML specifications is presented in [5]. Preconditions are used to identify the allowed ranges and boundary values of input parameters of method calls, while the postconditions are used to evaluate the results of the call, forming a test oracle. (Test oracles are separate evaluation units for go/no go test.)

Test generation based on constraint languages (e.g. Object Constraint Language [1]) is also a promising approach. OCL is a formal language used to describe constraints on UML models. The OCL expressions may be invariant conditions that must hold for the system or multiplicity constraints over objects described in a model. OCL expressions are used to specify pre- and postconditions of method calls as well. This paper describes an idea aiming at robustness testing of a software based on UML models, extended by OCL constraints.

Robustness is the degree to which the system operates correctly in the presence of exceptional inputs and stressful conditions. In case of robustness testing, test cases contain unexpected input parameters for the system under test and it is examined whether the response of the system is acceptable (proper error codes are returned without crash, restart, omission). Examining results obtained from such experiments, robustness of software under test can be estimated.

II. Information from OCL Constraints

Using UML models in test generation, all the necessary information to call an operation with appropriate parameters can be extracted. By using OCL constraints, all additional information can be extracted from the model, like the acceptable range of a parameter's value, and boundary values as well.

For example, if the range for an input integer value i is defined as positive and less then 100, possible values for robustness testing are obviously $i < 0$ or $i > 100$.

OCL postconditions specify what have to be guaranteed after the operation returns. Postconditions

not only define a range for the return value, but can determine several other conditions that must hold after the operation returns. In case of test generation, postconditions carry the necessary information to assess the results of operation calls.

Using the abstract syntax tree (AST) of the OCL conditions, it is possible to separate the different conditions from each other along and and or keywords, and check each subtree separately. Then applying Depth First Search, parameters can be evaluated according to their operators. This strategy is used both in case of preconditions (to extract input parameter values) and postconditions (to check return values).

III. Tool Support

The tool supporting test generation based on OCL constraints is currently under development. It is implemented as an Eclipse plug-in, hence it is able to work on UML models created within Eclipse. The tool works on applications written in Java, since it uses Javassist [2] byte code manipulator to load classes and execute operation calls.

With the help of the Javassist library, the plug-in is able to find the class files containing the operations to test (from the set of appointed class files). The plug-in has a blank $TestExecutor$ class file, which is modified by reflective programming. This modification is done in order to create and load the objects to test, and call their operations with the appropriate parameters (see Figure 1). These latter steps are hidden from the user, executed automatically in the background.

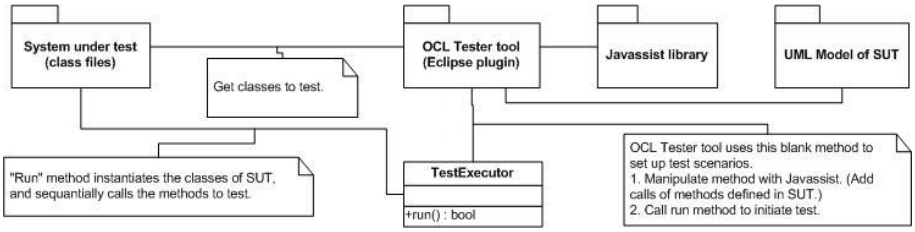

Figure 1: Test method instrumentation

IV. Conclusion and Future Work

In this paper one possible use of OCL constraints in robustness testing was introduced. The tool using this approach is under development, supporting a subset of the OCL. Further research is needed to refine the selection of input parameters for the operations (combining exceptional inputs with random values, or using fitness functions for the selection), and to evaluate postconditions automatically. A promising complementary approach is to combine robustness testing supported by the tool with mutation of execution scenarios specified in the model.

References

[1] O. M. G. Inc., *Object Constraint Language – OMG Available Specification Version 2.0*, May 2006, URL: http://www.omg.org/cgi-bin/doc?formal/2006-05-01.

[2] S. Chiba, "Javassist: Java bytecode engineering made simple," *Java Developer's Journal*, Jan. 2004.

[3] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., *Model-Based Testing of Reactive Systems*, Springer Berlin/Heidelberg, 2005.

[4] G. T. Leavens and Y. Cheon, *Design by Contract with JML*, Sept. 2006, URL: http://www.eecs.ucf.edu/ leavens/JML//jmldbc.pdf.

[5] F. Bouquet, F. Dadeau, and B. Legeard, "Automated boundary test generation from JML specifications," *Lecture Notes in Computer Science*, pp. 428–443, 2006, URL: http://www.springerlink.com/content/54q36j0r1362hnu5/.