Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 190 (2007) 21–32
www.elsevier.com/locate/entcs

Measuring a Java Test Suite Coverage Using JML Specifications

F. Dadeau[a,1,4] Y. Ledru[a,2] L. du Bousquet[a,3]

[a] *Laboratoire Informatique de Grenoble*
BP 72, 38402 Saint Martin d'Hères, France

Abstract

We propose in this paper a way to measure the coverage of a Java test suite by considering the JML specification associed to the Java program under test. This approach is based on extracting a predicate-based graph from the JML method specifications. We then measure the coverage of this latter w.r.t. nodes of the graph that are visited by the test suite. In addition, we propose to check whether the test suite satisfies classical condition coverage criteria. We also introduce a tool, to be used as precompiler for Java, that is in charge of measuring and reporting the coverage according to these criteria.

Keywords: Specification coverage, test suite, Java, JML, condition coverage.

1 Introduction

The essence of testing consists in executing the system under test in order to find bugs [21]. Nevertheless, testing can not be a complete approach since exhaustive testing is not applicable; the validation engineer is often left with a test suite that did not detect any bug in the program. How can he/she be sure that the test suite that was run is relevant enough to be confident in the program? One solution is to evaluate the quality of the test suite.

Several works on test suite evaluation exist, such as exercising the test suite on mutations of a program. The most relevant technique is to measure the coverage of the test suite. Usually, the coverage is measured on the control-flow graph of the program [21], or on the data-flow of the program [24]. In addition, a specification coverage can possibly be measured.

[1] Email: Frederic.Dadeau@imag.fr
[2] Email: Yves.Ledru@imag.fr
[3] Email: Lydie.du-Bousquet@imag.fr
[4] This work has been partially funded by the RNTL POSE project (ANR-05-RNTL-01001), supported by the French Ministry of Research and New Technologies.

doi:10.1016/j.entcs.2007.08.003

The recent arise of annotation languages makes it possible to specify the behavior of programs (i.e. of methods) inside the source code in terms of pre- and postconditions. It provides another "vision" of what a method should do, which can also be seen as expressing low-level *requirements*. It also provides a black-box view of what a method should do, expressed in terms of a *contract* [20].

Model-based testing [2] consists in computing test suites from a model of the considered program or system. Model-based conformance testing consists in ensuring that the program does not have an unintended behavior w.r.t. its specification. This conformance can be observed through *observation* points or using a run-time assertion checking mechanism if the proximity of both the specification and the program makes it possible. In this context, the Java Modeling Language [16] (JML) has been introduced to act as a behavioral interface specification language (BISL) for Java programs. JML can be used as an oracle for testing, considering that if no JML assertion is ever violated during the program execution, then the test succeeds, otherwise, it fails.

A previous work [3] has introduced the principles of model-based testing from JML specifications. Thus, some JML-based coverage criteria were used to guide the test target definition. We came to the idea that the coverage criteria defined in the latter work could be used to evaluate test suites that would have been produced by several tools, using different approaches, such as combinatorial testing (e.g. JMLUnit [6], Tobias [18]) or random testing (e.g. Jartege [23]).

We propose an approach for evaluating test suites for Java programs w.r.t. the coverage of an associated JML specification, expressing the behavior and/or the requirements of the methods. In addition, we propose to check different condition coverage criteria, that are contained within the disjunctions of the predicates of the specification.

The paper is organized as follows. Section 2 introduces the modeling possibilities provided by the Java Modeling Language. The first coverage criterion, based on the method specifications, is presented in Sect. 3. Section 4 is dedicated to the condition coverage definition. The principles of the measure and especially the implementation and the experiments are detailed in Sect. 5. Section 6 presents the related work, before concluding and providing a glimpse of the future work in Sect. 7.

2 Java Modeling Language

The Java Modeling Language –JML– has been designed by Gary T. Leavens et al. at Iowa State University [16,17]. The modeling elements are displayed as annotations, embedded within the Java source code. JML is based on the design by contract principle (DBC) [20] which states that, at the invocation of a method the system has to fulfill a contract (by satisfying the method's precondition) and as a counterpart, the method is expected to fulfill its contract (by establishing its postcondition).

JML makes it possible to express the static part of the system, such as invariants, and the dynamic part of the system, using postconditions and history con-