ESC/Java2: Uniting ESC/Java and JML

Progress and Issues in Building and Using ESC/Java2, Including a Case Study Involving the Use of the Tool to Verify Portions of an Internet Voting Tally System

David R. Cok[1] and Joseph R. Kiniry[2]

[1] B65 MC01816
Eastman Kodak R & D Laboratories
Rochester, NY 14650-1816, USA
cok@frontiernet.net
[2] Department of Computer Science, University College Dublin,
Belfield, Dublin 4, Ireland*
kiniry@acm.org

Abstract. The ESC/Java tool was a lauded advance in effective static checking of realistic Java programs, but has become out-of-date with respect to Java and the Java Modeling Language (JML). The ESC/Java2 project, whose progress is described in this paper, builds on the final release of ESC/Java from DEC/SRC in several ways. It parses all of JML, thus can be used with the growing body of JML-annotated Java code; it has additional static checking capabilities; and it has been designed, constructed, and documented in such a way as to improve the tool's usability to both users and researchers. It is intended that ESC/Java2 be used for further research in, and larger-scale case studies of, annotation and verification, and for studies in programmer productivity that may result from its integration with other tools that work with JML and Java. The initial results of the first major use of ESC/Java2, that of the verification of parts of the tally subsystem of the Dutch Internet voting system are presented as well.

1 Introduction

The ESC/Java tool developed at DEC/SRC was a pioneering tool in the application of static program analysis and verification technology to annotated Java programs [13]. It was a successor to the ESC/Modula-3 tool [22], using many of the same ideas, but targeting a "mainstream" programming language. ESC/Java operates on full Java programs, not on special-purpose languages. It acts modularly on each method (as opposed to whole-program analysis), keeping the complexity low for industrial-sized programs, but requiring annotations on methods that are used by other methods. The program source and its specifications are translated into verification conditions; these are passed to a theorem

* Formerly with the Security of Systems Group at the University of Nijmegen.

G. Barthe et al. (Eds.): CASSIS 2004, LNCS 3362, pp. 108–128, 2005.

prover, which in turn either verifies that no problems are found or generates a counterexample indicating a potential bug. The tool and its built-in prover operate automatically with reasonable performance and need only program annotations against which to check a program's source code. The annotations needed are easily read, written and understood by those familiar with Java and are partially consistent with the syntax and semantics of the separate Java Modeling Language (JML) project [1, 19]. Consequently, the original ESC/Java (hereafter called ESC/Java) was a research success and was also successfully used by other groups for a variety of case studies (e.g., [16, 17]).

Its long-term utility, however, was lessened by a number of factors. First, as companies were bought and sold and research groups disbanded, there was no continuing development or support of ESC/Java, making it less useful as time went by. As a result of these marketplace changes, the tool was untouched for over two years and its source code was not available.

The problem of lack of support was further compounded because its match to JML was not complete, and JML continued to evolve as research on the needs of annotations for program checking advanced. This unavoidable divergence of specification languages made writing, verifying, and maintaining specifications of non-trivial APIs troublesome (as discussed in Section 5).

Additionally, JML has grown significantly in popularity. The activities of several groups [1, 3, 19, 27, 28] generated a number of tools that work with JML. Thus, many new research tools worked well with "modern" JML, but ESC/Java did not.

Finally, some of the deficiencies of the annotation language used by ESC/Java reduced the overall usability of the tool. For example, frame conditions were not checked, but errors in frame conditions could cause the prover to reach incorrect conclusions. Also, the annotation language lacked the ability to use methods in annotations, limiting the annotations to statements only about low-level representations.

The initial positive experience of ESC/Java inspired a vision for an industrial-strength tool that would also be useful for ongoing research in annotation and verification. Thus, when the source code for ESC/Java was made available, the authors of this paper began the ESC/Java2 project.

This effort has the following goals: (1) to make the source consistent with the current version of Java; (2) to fully parse the current version of JML and Java; (3) to check as much of the JML annotation language as feasible, consistent with the original engineering goals of ESC/Java (usability at the expense of full completeness and soundness); (4) to package the tool in a way that enables easy application in a variety of environments, consistent with the licensing provisions of the source code release; and (5) as a long-term goal, and if appropriate, to update the related tools that use the same code base (Calvin, RCC, and Houdini [12]) and to integrate with other JML-based tools. This integration will enable testing the tool's utility in improving programmer productivity on significant bodies of Java source; the tool will also serve as a basis for research in unexplored aspects of annotation and static program analysis.