Monitoring Java Code Using ConGu

Vasco T. Vasconcelos, Isabel Nunes, and Antónia Lopes

Faculty of Sciences of the University of Lisbon, Campo Grande, 1749–016 Lisboa, Portugal,
{vv,in,mal}@di.fc.ul.pt

The formal specification of software components is an important activity within the task of software development, insofar as formal specifications are useful, on the one hand, to understand and reuse software and, on the other hand, to test implementations for correctness.

Design by Contract (DBC) [10] is widely used for the specification of object-oriented software. There are a number of languages and tools (e.g., [3,4,8,9]) that allow equipping classes and methods with invariants, pre and post-conditions, which can be monitored for violations at runtime. In the DBC approach, specifications are class interfaces (Java interfaces, Eiffel abstract classes, etc) annotated with contracts expressed in a particular assertion language, which is usually an extension of the language of boolean expressions of the OO language.

To build contracts using these languages one must observe the following: *(i)* contracts are built from boolean assertions, thus procedures (methods that do not return values) cannot be used; *(ii)* contracts should refer only to the public features of the class because client classes must be able not only to understand contracts, but also to invoke operations that are referred to in them—e.g., clients must be able to test pre-conditions; *(iii)* to be monitorable, a contract cannot have side effects, thus it cannot invoke methods that modify the state. These restrictions bring severe limitations to the kind of properties we can express directly through contracts. Unless we define a number of, otherwise dispensable, additional methods, we are left with very poor specifications.

Model-based approaches to DBC, like those proposed for Z [12], Larch [6], JML [9], and AsmL [2], overcome these limitations by specifying the behavior of a class, not via the methods available in the class, but else through very abstract implementations based on basic elements available in the adopted specification language. Rather than a *model based* approach, we instead adopted a *property based* algebraic approach to specifications, motivated and described in reference [11].

ConGu (Contract Guided System Development [5]) is a project whose aim is the development of a framework to create property-driven algebraic specifications and to fully test Java implementations against them. We find it important to equip property-driven approaches with tools similar to the ones currently available for model-driven approaches. Support for checking implementations against algebraic specifications is, as far as we know, restricted to a few approaches (cf [1,7]), which have limitations our approach overcomes.

The key idea of the **ConGu** approach is to reduce the problem of testing implementations against algebraic specifications to the runtime monitoring of contract annotated classes, which are automatically generated. Runtime contract monitoring is supported today by several runtime assertion-checking tools.

The **ConGu** main components are specifications, modules, and refinements. The *specifications* we use in this context are algebraic, property-driven insofar as they define sorts and operations on those sorts. In general terms, **ConGu** supports partial specifications—whose operations can be interpreted by partial functions—with conditional axioms. Each specification defines a single sort but it may use other sorts while defining, for example, parameters or results of operations. Specifications with external references to other sorts or operations are meaningful only when they are put together with the specifications that define all those references. We use the notion of *module* to denote the set of specifications that, together, are self-contained.

In order to check the behavior of Java classes against specifications—violations of an axiom or a domain restriction—the gap between specifications and Java classes must be bridged. For this purpose, *refinement mappings* have to be defined indicating which sort is implemented by which class, and which operation is implemented by which method. Because this activity does not require any knowledge about the concrete representation, refinement mappings are quite simple to define.

In this presentation we put forward an overview of the **ConGu** framework and demonstrate the **ConGu** tool, implemented as a plugin for the Eclipse IDE. The tool allows users to test Java classes—no source code needed, just bytecode—against a module of specifications, and to discover runtime axiom violations. It reads algebraic specifications and a mapping relating specifications and Java entities, and generates a number of classes that are used to test the original implementation against the given specifications, in a way that is transparent to the user. The technique used by **ConGu** surpasses the above referred limitations in what contracts are concerned: all specification properties are checked against implementations because monitorable contracts are generated that cover them all. We also report on the use of the **ConGu** tool in the context of an undergraduate programming course.

References

1. S. Antoy and R. Hamlet. Automatically checking an implementation against its formal specification. *IEEE Transactions on Software Engineering*, 26(1):55–69, 2000.
2. M. Barnett and W. Schulte. Spying on components: A runtime verification technique. In *Workshop on Specification and Verification of Component-Based Systems*, 2001. Published as Iowa State Technical Report 01-09a.
3. Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, volume 3362 of *LNCS*, pages 49–69. Springer, 2004.
4. D. Bartetzko, C. Fisher, M. Möller, and H. Wehrheim. Jass, Java with assertions. In *Proceedings of the First Workshop on Runtime Verification*, volume 55(2) of *ENTCS*. Elsevier, 2001.
5. Congu: Monitoring Java code against algebraic specifications. `http://gloss.di.fc.ul.pt/congu/`.
6. John V. Guttag and James J. Horning. *Larch: Languages and Tools for Formal Specification*. Springer, 1993.
7. Johannes Henkel and Amer Diwan. A tool for writing and debugging algebraic specifications. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 449–458. IEEE Computer Society, 2004.

8. Rachel Henne-Wu, William Mitchell, and Cui Zhang. Support for design by contract in the C# programming language. *Journal of Object Technology*, 4(7):65–82, 2004.
9. JML: Java Modelling Language. `http://www.jmlspecs.org/`.
10. B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 2nd edition, 1997.
11. Isabel Nunes, Antónia Lopes, Vasco T. Vasconcelos, João Abreu, and Luís S. Reis. Checking the conformance of Java classes against algebraic specifications. In *Proceedings of the International Conference Formal Methods and Software Engineering*, volume 4260 of *LNCS*, pages 494–513. Springer, 2006.
12. J. M. Spivey. *The Z notation: a reference manual*. Prentice Hall, 1992.