# Position Paper: Opportunities and challenges for formal specification of Java programs

Joseph Kiniry and Erik Poll
Dept. of Computer Science, University of Nijmegen, The Netherlands
{kiniry,erikpoll}@cs.kun.nl

**Abstract**

This paper describes the main opportunities and challenges that we see for introducing more rigorous software engineering practices, particularly those centered on specification and validation, in industrial practice. Our perspective derives from our ongoing work on formal specification and verification of Java programs.

## 1 Introduction

The notion of trusted component has two primary needs: a *definition* and *domains of application*.

1. What does it mean for a component to be trusted? Is there a technical definition involving documentation and testing, or specifications and verifications? Does it involve corporations, trusted third parties, webs of trust and encryption?

   We believe that many of these ideas and approaches have a place in the domain of trusted components, but one key aspect that drives our work is highlighted by D.H. Lawrence when he said, "Never trust the artist. Trust the tale. The proper function of the critic is to save the tale from the artist who created it."

   To rephrase in this setting, we must understand what a component does and verify (as critics) that it does what it is supposed to. We must not blindly trust the source from which it comes. This means that, in the least, we need a language in which to (a) express the rich properties of reusable components, and (b) a means by which to verify those properties.

2. As is evidenced by the state of the industry today, the vast bulk of software need not be "trusted" in any sense of the term. Software is not well documented enough, in the general, to understand what it is supposed to do even in the most gross sense.

   This means we must identify a domain that needs trusted components "yesterday", not in ten years. This need will ensure that the transition from research to development happens at a reasonable pace as businesses do not typically change their behavior until the financial picture dictates that they do so.

In the LOOP group at the University of Nijmegen we have been working on the formal specification and verification of Java programs for several years. The specification language we use is the Java Modeling Language, or JML, and the

main application area for our work is Java Card, the dialect of Java for programming smart cards. So for us, a trusted component is simply a well-specified Java class. Examples include a smart card program or an API class provided by a smart card operating system.

Section 2 describes this work in more detail, and then Section 3 describes what we believe are the most important opportunities and challenges in this field, especially with regards to the issues of trusted components.

# 2 Background

### Verification of Java programs

Our work in formal specification and verification of Java programs began with the formalisation of a denotational semantics of sequential Java. This work uses a coalgebraic semantics formalised in the theorem provers PVS and Isabelle. We have also developed an associated tool, the LOOP tool, that, given a Java program, returns a description of its semantics. Our formalisation does not include Java threads, but apart from that, we try to cover all the complications of real Java. For example, our formalisation includes exceptions, (possibly labeled) breaks and continues, arrays, inheritance (including overloading, overriding, and hiding), static fields and methods, static initialisation, etc.

This formalisation has since been extended to include the semantics of JML. Consequently, the LOOP tool has also been extended. It now takes a Java program with JML annotations as input and returns the aforementioned semantics of the program along with a collection of proof obligations. To satisfy these obligations, we have developed two proof infrastructures for Java. The first is a Hoare logic; the second, a weakest precondition calculus. Both of them have been proven correct with respect to the underlying semantics.

Currently, work focuses on improving PVS support for automating the tedious work of verification as much as possible. Although we are very pleased with the progress of this work, as well as the size and complexity of programs we can now handle, the interactive verification of code remains an extremely labour-intensive and difficult task. So while our goal is fully automatic verification of Java programs that have high quality specifications, such work will remain the domain for specialists in academia for the foreseeable future.

For more information about this work, see `http://www.cs.kun.nl/ita/research/projects/loop`.

### Smart cards

All the examples that we work in the LOOP group are Java Card programs. Java Card is a "dialect" of Java developed for the programming of smart cards (see `http://java.sun.com/products/javacard`). It is essentially a subset of Java, with many features omitted from the language (e.g. reals, doubles, threads ...) and from the API, in light of the very limited resources and capabilities of a typical smart card. A few features have been added to provide more security than the standard Java sandbox, notably a firewall around every smart card application. Java Card is the leading language used in the newest generation of smart cards. These cards support the download of (digitally signed) code, and allow several applications to be installed on the same smart card.

**JML**

As mentioned earlier, the specification language we use is the Java Modeling Language (JML) [4]. JML is a behavioural specification language for Java. JML allows you to annotate Java source code with assertions that, among other things, specify preconditions, postconditions, or class invariants, to record detailed design decisions.

It is important to note that JML lets one specify more than just the state-centric contracts of many other approaches. Specification via Hoare triples and invariants is a fine first step, but many of the more subtle and necessary properties of trusted components cannot be easily specified with such a simple language. We need a more expressive language than first-order logic, a language that is "closer" to our domain. We believe that JML is an excellent early example of such a language for Java-based trusted components.

JML annotations are written as a special kind of Java comments which are ignored by a Java compiler but are processed by various tools that support JML. JML supports well-established notions in the specification of object-oriented language, also used in the "Design By Contract" approach pioneered in Eiffel, but is more expressive. For example, JML supports quantifiers `\forall` and `\exists`, specification-only variables, and "normal" as well as "exceptional" postconditions.

The initiative for the development of JML was taken by Gary Leavens at Iowa State University, but it is a very open effort that has quickly grown to attract many collaborators and contributors worldwide. In particular, there have been serious efforts to remove some of the incompatibilities between JML and the assertion language used for ESC/Java (see `http://research.compaq.com/SRC/esc/` for more information).

Several tools for JML are available; see `http://www.jmlspecs.org/`. q The most interesting for the typical developer is probably the runtime assertion checker which translates annotations into runtime checks [3].

# 3 Opportunities and Challenges

What follows is a summary of some of the main opportunities and obstacles—or, to phrase it positively, challenges—that we see in the field of formal specification for Java programs.

**Opportunity: JML**

We believe that JML presents an promising opportunity to introduce a formal specification language in industrial practice.

One important reason for this is that JML is easy to read for people who know Java. A central design decision for JML was to make the language familiar for non-experts. The language uses mainly Java syntax, thus is easy to understand for anyone familiar with Java. Like any assertion language where one adds annotations to code, JML is easy to use gradually, either in new projects or for legacy code, as one can simply begin by add just a few assertions to source code. The threshold to then use some tools, especially the runtime assertion checker, is also low. Code is going to be tested anyway, doing runtime assertion checking when testing takes little extra effort.

JML is an open project, and it is hoped that other groups working on specification languages or tools for Java will join in. Having a range of tools that support the same specification language could contribute to the success of JML, and to the success of these individual tools. Having to learn yet another specification language is often a major obstacle to using a new tool. If the academic community wants

their tools and methodologies to be accepted in industry, it is important to agree on some common syntax. The experience with UML shows this. There are over half a dozen runtime assertion checkers for Java, all using a slightly different syntax, which is a pity.

### Opportunity: Smart Cards

Smart cards present a great opportunity for the application of more rigorous approaches to software engineering and formal methods. Typical smart card applications are fairly small, the language and API used are relatively simple, and the applications of smart card obviously demand the highest possible levels of confidence in the correctness and security.

The smart card industry is more open to the use of new methodologies—including formal methods—than most industries. Large investments in formal specifications have been made to certify components of smart card operating systems. One issue that plays a role here is the relatively new ISO standard for the evaluation of information technology security, the so-called Common Criteria. Increasingly high levels of certification are being demanded for smart card applications, and for the highest levels of certification the Common Criteria requires the use of formal methods.

Of course, one should not expect people in industry to take the initiative to try out new tools, notations, or methodologies. To get them interested takes some hard work: you to go out and use an approach yourself to show them that it works on a realistic examples and is feasible in practice.

To convince people of the value of a formal specification language like JML, and the utility of the tool support that brings, several case studies for the domain of smart cards have been tackled by academia. For example, we have developed formal JML specifications for Java Card API [9, 10]; these specification have been used by others to specify and debug a realistic Java Card case study with ESC/Java [2]; the LOOP tool has been used to verify parts of this case study [1]; the runtime assertion checker for JML has been used to formally specify and debug an implementation of a potential future extension of the Java Card API [8].

### Challenge: Identifying and Specifying Security Properties

All the work on Java Card using JML above focuses on typical correctness properties, i.e. ensuring that code meets certain functional specifications. It is often not clear what the "security properties" are that people are really interested in, or how these could be conveniently specified – let alone verified. For example, implementations of Java use stack inspection to ensure some security properties, so maybe expressing properties of the stack is a conveniently way to specify security policies?

An interesting overview of security properties that are relevant for Java Card applications is given by Marlet and Le Metayer [5]. Many, but not all, of these properties are standard correctness properties that are easy to express as functional requirements.

### Challenge: Pointers

A large obstacle in our specification and verification work is dealing with pointer-like constructs like Java's references. Of course, Java is nicer than languages that allow pointer arithmetic when is comes to specification and verification. Still, the basic problems caused by aliasing remain: e.g. how to specify the absence of aliasing, how to ensure the absence of aliasing, or how to reason about code in the presence of aliasing. Of course, this is a very old problem. In fact, it is somewhat embarrassing to the field of academic research that this problem has been around for so long,

without any definitive solutions. This is not to say that there has not been good work on "alias control"; e.g., see [11], which also gives a good overview of earlier work on the topic and an illustration of the relevance to security, or [6, 7].

# References


[1] C.-B. Breunesse, B. Jacobs, and J. van den Berg. Specifying and verifying a decimal representation in Java for smart cards. In *9th Algebraic Methodology and Software Technology (AMAST)*, LNCS, St. Gilles les Bains, Reunion Island, France, September 2002. Springer-Verlag, Berlin.

[2] N. Cataño and M. Huisman. Formal specification of Gemplus's electronic purse case study. In L. H. Eriksson and P. A. Lindsay, editors, *Formal Methods: getting IT right – Formal Methods Europe (FME)*, volume LNCS 2391, pages 272 – 289, Copenhagen, Denmark, Jul 2002. Springer Verlag.

[3] Y. Cheon and G.T. Leavens. A Runtime Assertion Checker for the Java Modeling Language (JML). In H.R. Arabnia and Y. Mun, editors, *International Conference on Software Engineering Research and Practice (SERP '02), Las Vegas, Nevada*, pages 322–328. CSREA Press, 2002.

[4] G.T. Leavens, A.L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report 98-06q, Dep. of Comp. Sci., Iowa State Univ., 2002.

[5] R. Marlet and D. Le Metayer. Security properties and Java Card specificities to be studied in the SecSafe project. Technical Report SECSAFE-TL-006, Trusted Logic, August 23 2001. Available from `http://www.doc.ic.ac.uk/~siveroni/secsafe/docs.html`.

[6] P. Müller. *Modular Specification and Verification of Object-Oriented Programs*. PhD thesis, FernUniversität Hagen, 2001. Also as LNCS 2262, Springer-Verlag, 2002.

[7] P. Müller, A. Poetzsch-Heffter, and G.T. Leavens. Modular specification of frame properties in JML. *Concurrency and Computation: Practice and Experience*, To appear.

[8] E. Poll, P. Hartel, and E. de Jong. A Java reference model of transacted memory for smart cards. In *Fifth Smart Card Research and Advanced Application Conf. (CARDIS'2002)*. USENIX, Nov 2002. to appear.

[9] E. Poll, J. van den Berg, and B. Jacobs. Specification of the JavaCard API in JML. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, *Fourth Smart Card Research and Advanced Application Conference (CARDIS'2000)*, pages 135–154. Kluwer Acad. Publ., 2000.

[10] E. Poll, J. van den Berg, and B. Jacobs. Formal specification of the Java Card API in JML: the APDU class. *Computer Networks*, 36(4):407–421, 2001.

[11] J. Vitek and B. Bokowski. Confined Types in Java. *Computer Networks*, 36(4):407–421, 2001.