Formal Techniques for Java-Like Programs (FTfJP)*

Alessandro Coglio[1], Marieke Huisman[2],
Joseph R. Kiniry[3], Peter Müller[4], and Erik Poll[3]

[1] Kestrel Institute, USA
[2] INRIA Sophia-Antipolis, France
[3] Radboud University Nijmegen, The Netherlands
[4] ETH Zürich, Switzerland

Abstract. This report gives an overview of the sixth Workshop on Formal Techniques for Java-like Programs at ECOOP 2004. It explains the motivation for the a workshop and summarises the presentations and discussions.

1 Introduction

Formal techniques can help one analyse programming languages or individual programs and precisely describe and verify their properties. Languages such as Java and C# are interesting targets for the application of formal techniques, for a variety of reasons: their reasonably clear and standardised semantics, their type systems play a crucial role in guaranteeing security through type safety, and their novel paradigm for program deployment, which improves interactivity, portability and manageability, but also opens new possibilities for abuse and raises concern about security.

Work on formal techniques and tools for programming and on the formal underpinnings of the programming languages themselves complement each other. This workshop aims to bring together those people working on the formal underpinnings of, and those working on the formal techniques and tools for, programming Java-like languages. The topics covered thus include: language semantics, type systems, dynamic linking and loading, and specification and verification techniques.

The workshop was organized by Sophia Drossopoulou (Imperial College) Gary T. Leavens (Iowa State University), Peter Müller (ETH Zürich), Arnd Poetzsch-Heffter (Universität Kaiserslautern) and Erik Poll (Radboud University Nijmegen).

The selection of papers was done by a larger program committee, consisting of Armin Biere (ETH Zürich), John Boyland (University of Wisconsin), Alessandro Coglio (Kestrel Institute), Matthew Dwyer (Kansas State University), Susan

* The title of this report should be referenced as "Report from the ECOOP 2004 Workshop on Formal Techniques for Java-like Programs (FTfJP)".

J. Malenfant and B.M. Østvold (Eds.): ECOOP 2004, LNCS 3344, pp. 76–83, 2004.

Eisenbach (Imperial College), Michael Ernst (MIT), Marieke Huisman (INRIA Sophia-Antipolis), Joe Kiniry (Radboud University Nijmegen), Doug Lea (State University of New York at Oswego), Peter Müller (ETH Zürich), David Naumann (Stevens Institute of Technology), James Noble (Victoria University of Wellington), Erik Poll (Radboud University Nijmegen), and Wolfram Schulte (Microsoft Research).

Twenty-three persons attended this full-day workshop, 18 representing universities and academic research institutes, and 5 from industry. A complete list of participants is given at the end of this report. A number of other participants dropped by for specific presentations, to chat with particular speakers, etc.

2 Overview of the Presented Papers

Twenty position papers were submitted, of which eleven were accepted for presentation at the workshop. In a way it was disappointing to have such a high rejection rate for a workshop, but we deliberately chose for accepting fewer papers to make for a relaxed presentation schedule and allow plenty of time for discussion.

The position papers submitted were collected in an informal proceedings that appears as technical report, nr. NIII-R0426, Radboud University Nijmegen[1], 2004. This technical report, and the individual position papers, are available on the web via the FTfJP home page (`http://www.cs.ru.nl/~erikpoll/ftfjp`).

The topics addressed by the presented papers were:

- type systems for Java-like languages,
- Java implementation and compilation,
- program specification and verification,
- invariants, and
- ownership types.

There are connections between these topics, of course. The papers about type systems and Java implementation and compilation in a sense all revolve about the flexibility of Java-like languages, providing subtyping and dynamic class loading. There are connection between the last three topics. Invariants play a crucial role in specification and verification for OO languages, and ownership types are about ensuring encapsulation, which is clearly relevant in specification and verification, and is of particular importance in dealing with invariants.

3 Typing

The first session of the workshop was about to type systems, either for more flexible approaches for type checking, separate compilation and linking, or to provide more expressive type system that allow more code reuse.

[1] The University of Nijmegen was renamed Radboud University Nijmegen as of 1 September 2004.

Davide Ancona talked about joint research with Ferruccio Damiani, Sophia Drossopoulou, and Elena Zucca into maximising the possibilities for separate compilation. He presented a type system which allows compilation of an individual class in isolation, i.e. without any information available about other classes. The (polymorphic) type system infers the minimal assumptions on other classes needed to guarantee type correct of a class, as a set of constraints on type variables, allowing separately type-checked classes can be safely linked provided these assumptions are met. As discussed, this opens up several possibilities for compilation schemes that support stronger forms of separate compilation and allow a more selective and lightweight approach to recompilation.

Alex Buckley talked about joint work with Sophia Drossopoulou which took a further step in the line of research presented in Davide's talk, namely using type systems like the one discussed by Davide to allow a more flexible approach to dynamic linking. Rather than hard-coding type names in bytecode, as in current dynamic linking schemes for Java or C#, he considered the possibility of leaving type variables in bytecode, which can then lazily be instantiated to concrete types only at run time.

Christopher Anderson presented a different strain of work on type systems than the previous speakers: he was interested in more expressive type systems that provide programmers with more flexible patterns of code reuse. He presented The type system of the language Concord, which was joint work with Paul Jolly, Sophia Drossopoulou, and Klaus Ostermann. This work used a simple form of dependent types to express relationships between collections of classes called groups. This resulted in quite some discussion about the relation with other approaches, notably the use of virtual types in Beta. The main advantage of Concord over other attempts to use dependent types for similar purposes, notably Scala (`http://scala.epfl.ch/`) appears the simplicity of the language, with allowed of decidability of the type system and soundness to be proved.

4 Java Implementation and Compilation

The work presented by Alessandro Coglio and Neal Glew concerned the use of formal techniques to improve JVM implementation and Java compilation.

Alessandro Coglio talked about checking access to protected members in Java, one of the trickiest aspects of enforcing visibility (or access control) in Java. Indeed, as he had discovered, there are existing JVM implementations which accept incorrect programs and reject correct ones. He gave a very clear explanation of the rules involved, and presented an optimal strategy for checking protected access that demanded the minimal amount of rechecking when new classes are dynamically loaded. There was quite some discussion about why checking access to protected members is so tricky in the first place, which could be traced by to the possibility of subclasses being in different packages.

Neal Glew talked about joint work with Jens Palsberg about method inlining. Method inlining is a standard optimisation performed by compilers, but it can be invalidated by later class loading. He presented a framework for reasoning about

this, and presented a technique for inlining method invocations that could be proved sound using this framework. The idea of the technique is that at the moment of dynamically loading a class, a whole program analysis is done to inline methods calls in the loaded code if possible (devirtualisation), and to patch invalidated inlinings in all previously loaded code (revirtualisation).

There was some discussion about the way in which the dynamic class loading, which caused much of the complications in the topics discussed by Coglio and Glew, were also the inspiration for much of the work presented in the first session.

5 Invariants

Class invariants play a crucial role in any approach to specification and verification for object oriented code.

Cees Pierik talked about joint work with Dave Clark and Frank de Boer, about a special kind of invariant that seems to play an important role in some "creational" design patterns for OO programming. In particular, he was interested in design patterns that control the way in which objects of a certain class are created, such as the singleton or factory patterns. These invariants belong to a class rather than an individual object, and could thus be called static rather than instance invariants thereby quantifying over all objects of a given class.

Andreas Roth talked about joint work with Peter Schmitt in the KeY project into the possibility of verifying that Java programs satisfy invariants in a modular way. He began by demonstrating that in the presence of subtyping and aliasing, "local" verifications that each class satisfies its invariants do not suffice to guarantee that invariants are never broken. Andreas proposed a notion of module that would allow sound modular verification of invariants. His solution involved a notion of ownership, a topic which was further discussed in the last session of the workshop.

There was some discussion about the surprising fact that, although class invariants play such a central role in any approach to program specification and verification for OO languages, the precise meaning of the notion is so tricky. It clearly remains a very interesting topic for further research. Cees Pierik raised the possibility of using techniques developed in the concurrency community, more specifically the rely-guarantee approach, to tackle the problem of reasoning about invariants dealing with shared state.

6 Specification and Verification

There were two talks involving the specification languages Spec# (for C#) and JML (for Java) which are related in many ways. Both of these talks were about the use of method calls in specification, e.g. in pre- and post-conditions, as a natural and convenient abbreviation mechanism.

Typically, one insists that such methods are *pure*, meaning they should not have any side-effects. However, this is a very strong requirements that rules out many obviously harmless method calls from being used in specifications. Mike

Barnett talked about joint work with David A. Naumann, Wolfram Schulte, and Qi Sun, about extending the category of methods than can be used in specifications. He discussed a weaker notion of *observationally pure* and argued that methods used in specifications need only be observationally pure rather than pure. Mike gave some typical examples to show that many non-pure methods that one would like to use in specifications are indeed observationally pure, so that this is a very useful notion.

In the subsequent discussion Erik Poll noted that one would like to extend the category of methods that could be used in specifications even further, in particular to include the method that came up in the Singleton pattern in Cees Pierik's talk earlier, which does have a (clearly harmless) side effect the very first time it is called.

David Cok talked about his work on extending the program verification tool ESC/Java2 to support reasoning with specification that contained method calls. David considered some of the design decisions involved and raised some more fundamental design decisions about how to reason about the heap in verifying object oriented languages. This led to some discussion with Rustan Leino about how this is done in the new Boogie prover. David also expressed some dissatisfaction with the current notion of "purity" in JML, in line with the previous talks. However, he also noted that the possibility that pure methods can throw exceptions causes complications in reasoning about specifications containing method calls, and suggested that maybe the notion of (observationally) pure should be strengthened to disallow this.

7 Ownership Types

The last session of the workshop was devoted to ownership type systems, which continues to be a hot topic both at FTfJP and the main ECOOP conference.

Alex Potanin presented joint work with James Noble, Dave Clarke, and Robert Biddle. They developed a new type system, OGJ, which combines generic types and ownership [5]. Their type system supports the full type genericity of Generic Java as well as parametric ownership. The talk by Potanin focused on defaulting, a lightweight alternative to type inference. Defaulting allows one to integrate code that does not have ownership annotations. This approach is certainly relevant to all researchers in the area of ownership types, in particular, since type inference for ownership type systems does not yet produce practically useful results, as remarked by Jonathan Aldrich who was one of the several special visitors for this session.

Werner Dietl discussed joint work with Peter Müller on how exceptions can be supported on ownership type systems. He analysed four viable designs (1) cloning exception objects during propagation; (2) using unique references to transfer exceptions between contexts during propagation; (3) treating exceptions as global data; (4) handling exceptions by read-only references that may cross context boundaries. His presentation lead to an interesting discussion among the participants about the design principles of different ownership type systems, contrast-

ing parameterised type systems in the tradition of Clarke, Potter, and Noble's work [1] and Müller's more lightweight Universe type system [4].

This session on ownership type systems was effectively continued the next morning at the main ECOOP conference, where the papers presented in the first session after the invited talk were also about notions of ownership.

8 Discussion

At the end of the workshop there was some more general discussion about topics people wanted to raise then and there. The discussion quickly focused about the pros and cons of typing versus annotations as means of allowing more information to be expressed in programs, and possibilities of moving more annotations into type systems, the latter having the advantage of easier acceptance by programmers and offering more automation. Rustan Leino talked about efforts of moving 'non null'-ness, one of the most fundamental properties that crops up in any attempt to support code analysis or verification, into the type system of Spec#. Joe Kiniry raised the issue of overcoming programmer resistance to new type or annotation system and that a considerable amount of preparatory work, such as annotating APIs, was needed before one could begin to convince programmers of the usefulness. David Cok remarked that for this he would like to see a tool in the style of Daikon [2], which infers likely annotations by observing runtime behaviour of programs, that uses static checking instead of runtime verification; others observed that this is essentially what has been pursued in the Houdini work [3].

9 Conclusions

We were pleased to be able to announce at the workshop that two special journal issues dedicated to previous editions of the workshop had appeared in the month preceding this year's ECOOP. A special issue of the journal *Concurrency and Computation: Practice and Experience (CCPE)* appeared about FTfJP'2002 (CCPE, Volume 16, Issue 7, 2004), and a special issue of the online *Journal of Object Technology (JOT)*, appeared about FTfJP'03 (JOT (`http://www.jot.fm`), Volume 3, Number 6, 2004). The fact that these issues appeared almost simultaneously proves one clear advantage of online only journals! There are plans for a special issue of JOT about FTfJP'2004, for which Joe Kiniry and Susan Eisenbach have volunteered to serve as editors.

Even though this is now the sixth workshop in the series, the workshop is still going strong. The focus of the workshop has shifted somewhat over time, as different topics become more or less popular, or essentially resolved. For instance, it was interesting to note that there was not a single presentation about bytecode verification this year. It is nice to observe that the workshop has helped in raising some interesting topics for research, with some papers addressing issues raised at earlier editions of the workshop, and to observe the way it has contributed

to fostering collaborations, all of which has resulted in good work presented not just at this workshop but also at the main ECOOP conference.

The workshop has somewhat outgrown the standard workshop format, given the number and quality of submissions it typically received, and the number of people that want to participate. But the interest it generates and the audience it attracts proves that it clearly serves a useful purpose and we look forward to organising another FTfJP workshop at next year's ECOOP.

List of Participants

- Davide Ancona (University of Genua, Italy)
- Christopher Anderson (Imperial College, UK)
- Mike Barnett (Microsoft Research, USA)
- Alex Buckley (Imperial College, UK)
- Alessandro Coglio (Kestrel Institute, USA)
- David Cok (Kodak Eastman, USA)
- Adam Darvas (ETH Zürich, Switzerland)
- Werner Dietl (ETH Zürich, Switzerland)
- John Field (IBM's T. J. Watson Research Center, USA)
- Neal Glew (Intel, USA)
- Johan Glimming (KTH, Sweden)
- Marieke Huisman (INRIA Sophia-Antipolis, France)
- Bart Jacobs (K.U. Leuven, Belgium)
- Joe Kiniry (Radboud University Nijmegen, Netherlands)
- Peter Müller (ETH Zürich, Switzerland)
- Rustan Leino (Microsoft Research, USA)
- Cees Pierik (University of Utrecht, Netherlands)
- Frank Piessens (K.U. Leuven, Belgium)
- Alex Potanin (Victoria University of Wellington, New Zealand)
- Erik Poll (Radboud University Nijmegen, Netherlands)
- Andreas Roth (University of Karlsruhe, Germany)
- Mirko Viroli (University of Bologna, Italy)
- Joe Zhou (University of Leicester, UK)

References

1. D. G. Clarke, J. M. Potter, and J. Noble. Ownership types for flexible alias protection. In *Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 33(10) of *ACM SIGPLAN Notices*, October 1998.
2. Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):1–25, 2001.
3. Cormac Flanagan and K. Rustan M. Leino. Houdini, an annotation assistant for ESC/Java. In J. N. Oliveira and P. Zave, editors, *FME 2001*, volume LNCS 2021, pages 500–517. Springer, 2001.

4. P. Müller. *Modular Specification and Verification of Object-Oriented Programs*, volume 2262 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
5. A. Potanin, J. Noble, D. Clarke, and R. Biddle. Generic ownership. Technical Report CS-TR-03-16, Victoria University of Wellington, 2003.