abc: The AspectBench Compiler for AspectJ

A Workbench for Aspect-Oriented Programming Language and Compilers Research *

Chris Allan[1], Pavel Avgustinov[1], Aske Simon Christensen[2], Bruno Dufour[3], Christopher Goard[3], Laurie Hendren[3], Sascha Kuzins[1], Jennifer Lhoták[3], Ondřej Lhoták[3], Oege de Moor[1], Damien Sereni[1], Ganesh Sittampalam[1], Julian Tibble[1], Clark Verbrugge[3]

[1] Programming Tools Group, Oxford University, United Kingdom
[2] BRICS, University of Aarhus, Denmark
[3] Sable Research Group, McGill University, Montreal, Canada

http://aspectbench.org

Abstract

Aspect-oriented programming (AOP) is gaining popularity as a new way of modularising cross-cutting concerns. The *aspectbench compiler* (*abc*) is a new workbench for AOP research which provides an extensible research framework for both new language features and new compiler optimisations. This poster presents the *abc* architecture, as well as example uses.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—compilers

General Terms Performance, Experimentation, Languages

Keywords AspectJ, aspect-oriented programming language, optimizations, language extensions

1. INTRODUCTION

AspectJ is a seamless extension of Java for expressing program features that *crosscut* Java's abstraction mechanisms [10]. AspectJ compilers take as input both ordinary Java classes/interfaces and AspectJ aspects, producing pure Java bytecode as output. Aspects contain advice declarations, each composed of a *pointcut* (a pattern matching runtime events, or *join points*) and an *advice body* (what to do at matching join points). An AspectJ compiler must match all program points corresponding to the pointcuts and weave in instructions to invoke methods corresponding to the advice bodies. Sometimes matching cannot be completed at compile-time, so runtime checks must be compiled into the woven code [9, 11].

The original *ajc* compiler, originally developed at Xerox Parc and now supported by the Eclipse project (`eclipse.org/aspectj`), was designed to have good compile-time speed, to support incremental compilation, and to integrate well with Eclipse IDE tools. While *ajc* meets its goals well, it does not provide a very suitable platform for research into language extensions and compiler optimisations.

The *aspectbench* compiler, or *abc* for short, was developed with two main goals: (1) to provide a convenient platform for experimenting with new language extensions, and (2) as a toolkit for developing new optimisations to reduce the costs of aspect-related overheads.

* This work was supported, in part, by EPSRC, NSERC and IBM.

OOPLSA'05, October 16–20, 2005, San Diego, California, USA.

2. ARCHITECTURE

In order to provide both extensibilty and optimisation, *abc*'s frontend has been based on Polyglot [12], and its backend on Soot [15]. Polyglot provides excellent support for language extensions while Soot provides considerable infrastructure for program analysis and optimisation.

The overall architecture of *abc* is given in Figure 1. The frontend first processes the Java and AspectJ inputs, and splits the representation into a pure Java AST and a separate data structure containing all of the aspect-specific information (*AspectInfo*). The backend then generates the low-level Jimple IR for the pure Java part and then weaves the aspects in based on the information stored in *AspectInfo*. Finally, an optimisation pass is performed, producing optimised Jimple which can then be translated to Java bytecode or decompiled back to pure Java source. The analysis of woven bytecode may result in information which can improve weaving decisions. As indicated by the dashed line in Figure 1, information from this analysis can be fed back to the weaver for reweaving, which can result in further optimisations.

3. LANGUAGE EXTENSIONS

The *abc* workbench supports new language extensions at several levels including syntax, type systems, and new kinds of join point [3]. We have used *abc* to extend AspectJ to support trace matching with free variables [1]. This is a non-trivial extension which supports matching traces rather than individual join points and allows the programmer to track the behaviour of individual objects via variable binding. It is this mechanism for variable binding that is the main innovation over earlier proposals [5, 16].

Many other research groups are also using *abc* for interesting extensions including: *LoopsAJ*, an extension for loop join points for use in scientific computing [8]; *SCoPE*, an extension that supports static conditional pointcut evaluation [2]; *J-LO*, an extension for supporting the dynamic checking of temporal properties [14]; and *Cona*, an extension which supports contracts for AspectJ [13].

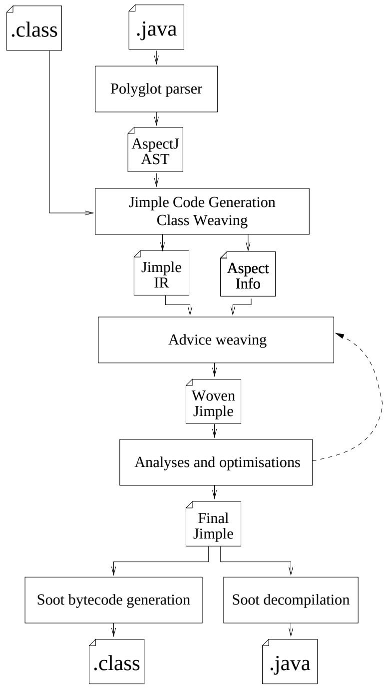

Figure 1. *abc* architecture

4. OPTIMISATIONS

Before we started working on *abc*, there had not been much emphasis on measuring or reducing overheads due to aspect weaving. In order to determine what sorts of overheads were involved, we performed a study on a variety of AspectJ benchmarks [7]. In that study we built a modified version of *ajc* which annotates bytecode instructions with different tags specifying whether the instruction is overhead, and if so, what kind of overhead. We then used a specialised version of the *J dynamic metrics tool [6] to calculate various AspectJ-specific dynamic metrics. This study showed that there were significant overheads, particularly for **around** advice and **cflow** pointcuts.

In developing *abc*, we tackled these overhead problems by developing new strategies for **around** weaving and by applying both intra- and inter- procedural analysis for minimising and eliminating **cflow** overheads [4]. These optimisations can have very significant impacts on peformance, resulting in order of magnitude speed improvements for some benchmarks.

5. CONCLUSIONS AND FUTURE WORK

Language design and optimisation strategies for aspect-oriented programming languages are becoming important research areas. This poster gives an overview of the *abc* workbench which has been designed for language research, by supporting easy extensiblity, and compiler research, by providing support for sophisticated analysis and optimisations.

The *abc* workbench has been used successfully by our group and many other research groups. We intend to continue to use *abc* to experiment with new language designs and optimisations, and also to further improve the workbench based on our experiences and feedback from our users.

References

[1] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to AspectJ. In *Object-Oriented Programming, Systems, Languages and Applications*, pages 117–128. ACM Press, 2005.

[2] Tomoyuki Aotani and Hidehiko Masuhara. Compiling conditional pointcuts for user-level semantic pointcuts. In *Proceedings of the SPLAT workshop at AOSD 2005*, 2005.

[3] Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. *abc*: An extensible AspectJ compiler. In *Aspect-Oriented Software Development (AOSD)*, pages 87–98. ACM Press, 2005.

[4] Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Optimising AspectJ. In *Programming Language Design and Implementation (PLDI)*, pages 117–128. ACM Press, 2005.

[5] Remi Douence, Pascal Fradet, and Mario Südholt. Trace-based aspects. In *Aspect-oriented Software Development*, pages 141–150. Addison-Wesley, 2004.

[6] Bruno Dufour, Karel Driesen, Laurie Hendren, and Clark Verbrugge. Dynamic metrics for Java. In *Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications*, pages 149–168. ACM Press, 2003.

[7] Bruno Dufour, Christopher Goard, Laurie Hendren, Oege de Moor, Ganesh Sittampalam, and Clark Verbrugge. Measuring the dynamic behaviour of AspectJ programs. In *19th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2004)*, pages 150–169, 2004.

[8] Bruno Harbulot and John R. Gurd. Using AspectJ to separate concerns in parallel scientific Java code. In *Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 122–131. ACM Press, 2004.

[9] Erik Hilsdale and Jim Hugunin. Advice weaving in AspectJ. In K. Lieberherr, editor, *Aspect-oriented Software Development (AOSD 2004)*. ACM Press, 2004.

[10] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In J. Lindskov Knudsen, editor, *European Conference on Object-oriented Programming*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[11] Hidehiko Masuhara, Gregor Kiczales, and Chris Dutchyn. A compilation and optimization model for aspect-oriented programs. In *Compiler Construction*, volume 2622 of *Springer Lecture Notes in Computer Science*, pages 46–60, 2003.

[12] Nathaniel Nystrom, Michael R. Clarkson, and Andrew C. Myers. Polyglot: An extensible compiler framework for Java. In *12th International Conference on Compiler Construction*, volume 2622 of *Lecture Notes in Computer Science*, pages 138–152, 2003.

[13] Therapon Skotiniotis and David H. Lorenz. Cona: aspects for contracts and contracts for aspects. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 196–197, New York, NY, USA, 2004. ACM Press.

[14] Volker Stolz and Eric Bodden. Temporal Assertions using AspectJ. In *Fifth Workshop on Runtime Verification (RV'05)*, Electronic Notes in Theoretical Computer Science, Edinburgh, Scotland, UK, 2005. Elsevier Science Publishers.

[15] Raja Vallée-Rai, Etienne Gagnon, Laurie J. Hendren, Patrick Lam, Patrice Pominville, and Vijay Sundaresan. Optimizing Java bytecode using the Soot framework: Is it feasible? In *Compiler Construction, 9th International Conference (CC 2000)*, pages 18–34, 2000.

[16] Robert Walker and Kevin Viggers. Implementing protocols via declarative event patterns. In *ACM Sigsoft International Symposium on Foundations of Software Engineering (FSE-12)*, pages 159–169, 2004.