AOSD 2006 Tutorial:
abc : How to implement your own tools for AOP research

October 14, 2005

1 Instructors

1.1 Oege de Moor

Contact Information:

Professor Oege de Moor,
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
`oege@comlab.ox.ac.uk`
`http://www.comlab.ox.ac.uk/oucl/work/Oege.de.Moor/`

Brief Biography: Oege de Moor received an M.Sc. degree from the University of Utrecht, the Netherlands. He did his doctoral (D.Phil.) work at Oxford, on a characterisation in category theory of algorithmic paradigms such as dynamic programming. He has held visiting appointments at the University of Tokyo, Chalmers University and Microsoft Research (both Redmond and Cambridge).

Since 1994 he has been a faculty member at Oxford, where he founded the Algebra of Programming research group, jointly with Richard Bird. After the publication of a textbook on that subject in 1997, he turned to the general area of meta-programming, and founded the Programming Tools Group. Research topics in the group include type systems for metaprogramming, mechanised support for refactoring, and aspect-orientation. During the 2003-2004 academic year Professor Hendren visited Oxford University, and they started the joint *abc* project, the topic of this tutorial.

Summary of Teaching Experience: Oege de Moor has taught at Oxford for 11 years, on topics ranging from computational geometry to database systems. He has also lectured at specialist summer schools in Brazil, Denmark, Germany, the Netherlands and Portugal. For the past two years he has taught a a course at Oxford that focuses on code queries, refactoring and aspect-oriented programming.

Primary contact: yes

1.2 Laurie Hendren

Contact Information:

Professor Laurie Hendren,
Professor, School of Computer Science, McGill University

3480 University Street, Montreal, Quebec, Canada H3A 2A7
hendren@cs.mcgill.ca
http://www.sable.mcgill.ca/~hendren

Brief Biography: Laurie Hendren received the B.Sc.(Honours) and M.Sc degrees in Computing and Information Science at Queen's University, Kingston, Canada. She received the Ph.D. degree from Cornell University, Ithaca, N.Y. Her Ph.D. thesis was in the area of automatic parallelization of programs with pointer data structures.

Since 1990 she has been a faculty member in the School of Computer Science at McGill University where she has led the McCAT optimizing/parallelizing compiler project, and currently leads the Sable project that concentrates on compiling and optimizing Java. During the 2003-2004 academic year she visited Professor de Moor at Oxford University, where they started the joint *abc* project, the topic of this tutorial.

Summary of Teaching Experience: Professor Hendren has had substantial teaching experience, 15 years teaching at McGill where she has taught courses on Java, compiler design, and advanced compiler topics. Since 2006, she has been teaching a course on advanced object-oriented and aspect-oriented programming. Detailed course web pages are available from her web site. She has twice been nominated for the award as best teacher in the Faculty of Science at McGill.

Primary contact: no (see Oege de Moor, above)

1.3 Ondřej Lhoták

Contact Information:

Professor Ondřej Lhoták,
Assistant Professor, School of Computer Science, University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1
olhotak@uwaterloo.ca
http://plg.uwaterloo.ca/~olhotak

Brief Biography: Ondřej Lhoták received a B.Math. (Honours) degree in Computer Science from the University of Waterloo in Waterloo, Ontario, Canada. He received M.Sc. and Ph.D. degrees in Computer Science from McGill University in Montreal, Quebec, Canada. His thesis was in the area of interprocedural analysis of object-oriented and aspect-oriented programming languages.

Since January 2006, he is an assistant professor at the University of Waterloo. During his graduate studies at McGill University, he was a maintainer of the Soot framework on which *abc* is built. He was a member of the *abc* team from the beginning of the project, and continues to contribute.

Summary of Teaching Experience: Professor Lhoták has been a teaching assistant for courses on compilers, as well as introductory computer science courses. At PLDI 2003 and CASCON 2003, he was one of the presenters of a tutorial on the Soot framework.

Primary contact: no (see Oege de Moor, above)

1.4 Members of the abc team

In addition to the three primary instructors, at least five further members of the *abc* team will be on hand to assist with the tutorial: Pavel Avgustinov (Oxford), Eric Bodden (McGill), Elnar Hajiyev (Oxford), Neil Ongkingco (Oxford), and Julian Tibble (Oxford).

2 Tutorial - Basic Information

Title: *abc*: how to build your own tools for AOP research

Length: whole day

Abstract: The proliferation of new features, analyses and optimisations for aspect-oriented programming languages necessitates a workbench for realistic experiments. The *AspectBench Compiler* (*abc* for short) provides such a workbench. The base compiler is a full implementation of the AspectJ language.

abc has been designed to disentangle the implementation of new, experimental features from the base compiler. *abc* has also been designed to enable the implementation of advanced analyses, for the purpose of optimisation, but also to enable static detection of bugs.

By working through concrete examples from the literature, participants will learn how to integrate their own language extensions, analyses and optimisations into *abc*.

Expected Audience: Researchers working on language features for aspect-oriented programming.

Level of Tutorial: intermediate.

Prerequistes for participants: Knowledge of Java and AspectJ.

Previous venues: This is an updated, expanded version of a tutorial at AOSD 2005.

Required equipment for presentation: PC projector, whiteboard.

Required equipment for participants: None.

Tutorial includes hands-on sessions: No - but hands-on exercises will be provided for completion outside the tutorial.

3 Tutorial — Synopsis

Topic and importance In recent years, there has been an explosion of proposals for new programming language features to support aspect-orientation. Along with the proposals for new features there have been calls to harness the new power of aspect-orientation through novel semantic analyses, for instance to support modular reasoning.

Often such proposals for new languages or analyses are supported by a semantic model, possibly implemented as a definitional interpreter. Eventually, however, new language features must be integrated into a full, industrial-strength programming language to prove their merits in practice. Such integration is particularly important to demonstrate the feasibility of an efficient implementation.

abc provides a workbench to conduct such experiments: it is a full implementation of AspectJ, it provides a highly extensible frontend, and an advanced analysis framework. It is furthermore designed to enable the effective implementation of optimisations, and indeed it produces better code than the only other available compiler for AspectJ, *ajc*.

Expected interest Extensions of AspectJ, as well as the analysis of aspect-oriented programs more generally, frequent topics at AOSD conferences. For example, at AOSD 2005 there were at least six papers in this category. In our view, this demonstrates that the topic of this tutorial is at the heart of ongoing research in the community, and we therefore expect considerable attendance from other research teams.

Even researchers who are not working directly on the topics of language extension and compilation will benefit by *abc*'s infrastructure. For example, research on aspect-mining or refactoring transformations needs a flexible frontend for a mature language, and *abc* provides that. Again, AOSD 2005 had a whole session devoted to refactoring tools.

Concrete evidence for the level of interest is provided by statistics on downloads of *abc* to date. Members of most major centres of AOSD research have downloaded a pre-release of *abc* and have told us they plan to evaluate it as a tool for their own research: British Columbia, Darmstadt, Duisburg/Essen, IBM Canada, Fukuoka, Lancaster, Manchester, Nantes, Northeastern (Boston), the University of Tokyo and Tokyo Institute of Technology, UCSD (San Diego) and the University of Washington (Seattle). Indeed, since it was released exactly one year ago, *abc* has been downloaded from 736 non-robot distinct IP addresses outside the authors' own institutions.

We believe our tutorial will also help attract more programming language and compiler researchers to the AOSD community. In summary, this tutorial will be attractive to a wide section of the AOSD research community, as well as programming language researchers who wish to enter that community.

Educational Goals Attendees will learn:

- the architecture of compilers for aspect-oriented languages, including common pitfalls
- how to implement their own new language features in AspectJ
- how to design and implement new analyses and optimisations for aspect-oriented language tools
- the pros and cons of a variety of building blocks for such tools, so participants can choose the best technology for their own research problem.

3.1 Synopsis and time schedule

The tutorial is in three parts:

- In the first part, we shall examine the frontend of *abc*, and how one adds new syntax and type checking rules. Originally the frontend was written using Polyglot, an extensible Java compiler from Cornell by Andrew Myers *et al.*. We shall illustrate the reasons why newer versions of *abc* will instead use JastAdd, an aspect-oriented compiler framework from Lund by Görel Hedin and Torbjörn Ekman. We shall present a number of example extensions, ranging from simple syntactic variations to fundamental changes.

- In the second part, we introduce the advice weaver of *abc*. We shall concentrate on its extension mechanisms, and show how one can easily introduce new kinds of joinpoint. The *abc* weaver uses the Soot bytecode analysis and transformation framework, and we present that in some detail, along with examples of new joinpoints. We conclude with a brief description of other features that require an extensible matcher, in particular features for information hiding.

- In the third part, we show how to integrate new program analyses into *abc*. After a brief overview of the basics of dataflow analysis, we show how intraprocedural analyses are used in *abc* and its extensions. In the final part of the tutorial, we sketch how interprocedural analyses can be used for aggressive optimisation and for checking complex properties of AspectJ programs.

Participants will be provided with practical, hands-on exercises that further illustrate the concepts covered in the lectures. No time is scheduled for these exercises during the six tutorial hours, but we expect that participants may try them either later or during the breaks.

The following is a more detailed synopsis:

Overview Aims of this tutorial. The design goals of *abc*. The architecture of aspect-oriented compilers. How different design goals led to differences between *abc* and *ajc*.

Frontend extension Syntax. Lexing AspectJ: the need for a stateful lexer. An LALR(1) grammar for AspectJ. Reasons for not choosing a scannerless GLR parser. Semantic checking. Desiderata for extensible semantic checking: why we are changing to JastAdd. Case study: semantic checks for tracematches.

Advice weaving The advice weaver of *abc*. The Jimple intermediate representation. Finding join point shadows. Normalising pointcuts into a regularised intermediate representation. Matching pointcuts against shadows. The weaving instructions. Case study: array set and get joinpoints. Modifying the matcher for open modules.

Analysis and optimisations Brief introduction to program analysis techniques. Intraprocedural analyses with Soot in *abc*. Case study: find loop joinpoints (from Harbulot's loopAJ). Interprocedural optimisation: eliminating the runtime overheads of *cflow*. The Paddle framework. Checking aspects for purity.

3.2 Links to materials

This is an overhaul of a previous tutorial, and the materials are still actively under development. The main reference to demonstrate maturity of the material is the *abc* website itself, in particular its publications section:

`http://aspectbench.org`

The *abc* team has given a very similar tutorial at AOSD 2005:

`http://abc.comlab.ox.ac.uk/documents/abcTutorialNotes.pdf`

That tutorial had about 12 participants, although a considerable number did not officially register.

This new tutorial different in a number of ways:

- *abc* is about to undergo a fundamental change, replacing Polyglot in the frontend by JastAdd. This tutorial will contrast the two technologies, providing valuable insight for other researchers that are shopping around for an extensible compiler infrastructure.
- The example extensions in this tutorial are far more mature, reflecting a year's experience with *abc*, by the authors and by third parties. Among the example extensions that will be discussed are Aldrich's open modules and history-based advice.
- The analysis framework is covered in much greater depth, filling in some specialist knowledge about program analysis for researchers whose background is not in compilers research per se. The examples of analysis not only cover optimisation, but also its use in defining interesting new joinpoint types (such as Harbulot's loop joinpoints), and in proving complex properties of aspect-oriented programs (such as aspect purity).
- The new tutorial is twice as long as the old one, giving more time to cover both simple and advanced examples.

De Moor will run this tutorial for master's students at Oxford during the January - March term. The materials will therefore come as a complete course package, including class exercises and practical assignments, that participants can take away for home study, or indeed for teaching a similar course themselves.