Special Issue on Foundations of Aspect-Oriented Programming

— The guest editors' introduction —

Pascal Fradet

INRIA Rhône-Alpes, 655 av. de l'Europe, 38330 Montbonnot, France

Ralf Lämmel

Microsoft, One Microsoft Way, 98052 Redmond, WA, USA

Abstract

This SCP special collects articles that make contributions to the foundations of aspect-oriented programming (AOP). Aspects have been developed over the last 10 years to facilitate the modularization of crosscutting concerns, i.e., concerns that crosscut with the primary modularization of a program. This special issue further continues the efforts of the annual FOAL workshop (Foundations of Aspect-Oriented Languages) in so far that it supports and integrates research on firm foundations of AOP. There are 5 contributions addressing the following issues: (i) a fundamental core language for aspects; (ii) subtleties of so-called around advice; (iii) aspects in higher-order languages; (iv) the interaction between aspects and generics; (v) a notion of aspects for reactive systems based on synchronous languages.

1 Preamble

Modularity concepts support the separation of concerns, and hence they are crucial for mastering the complexity of software. In recent years, the modularity concept of aspects and the notion of aspect-oriented programming have been introduced. Aspects facilitate the modularization of crosscutting concerns. Aspect-oriented programming (AOP) is part of a larger effort concerned with the complete life cycle and called Aspect Oriented Software Development (AOSD). AOP and AOSD have attracted a great deal of interest: various AOP tools are being developed and AOP

Email addresses: Pascal.Fradet@inria.fr (Pascal Fradet), Ralf.Lammel@microsoft.com (Ralf Lämmel).

21 June 2006

is now successfully used in real-world applications. However, a widely shared criticism of AOP is that it still lacks firm foundations. For instance, the untamed form of AOP that we see today is often criticized for defeating encapsulation and preventing modular reasoning. Likewise, the interference of multiple aspects is considered an unacceptable risk of current AOP techniques. Also, AOP language implementations tend to be ahead of programming-language theory. That is, actual language implementations end up providing expressiveness without well-defined static or dynamic semantics. Further, AOP has been primarily driven in a context that is biased towards mainstream, objected-oriented programming; the notion of aspects has been studied much less for languages other than Java and friends.

This special issue is dedicated to the foundations of aspect-oriented programming including formal methods, tools, languages and calculi, to the improvement of the state of the art of developing, understanding, controlling and reasoning about aspect-oriented programs. The special issue goes along with the successful series of FOAL workshops (Foundations of Aspect Oriented Languages). The call for papers mentioned the following themes of interest:

- Reasoning about aspect-oriented programs.
- Semantics of aspects and weaving.
- Formal aspect languages or calculi.
- Formal properties of aspects, advice, and join-point models.
- Type systems for aspect-oriented languages.
- Verification and static analysis of aspect-oriented programs.
- Theory of aspect composition and aspect interactions.
- Principled examples of aspect-oriented programming.

Most of these topics are addressed by the selected contributions.

2 Articles included in the special issue

We briefly summarize the selected articles and place them in the research context.

A Type-theoretic Interpretation of Pointcuts and Advice by *Jay Ligatti, David Walker and Steve Zdancewic*

This article is a properly extended version of the authors' ICFP 2004 paper ("A theory of aspects"), which by now may be considered a seminal paper in the research area of foundations of AOP. The article provides a semantical and type-system framework for AOP language constructs. The semantics of an idealized, user-friendly AOP language (MiniAML) is defined by giving a type-directed translation to a compact, well-defined core AOP language. The core language extends the simply-typed lambda calculus with explicitly labeled program points and first

class advice. These abstractions can be used both in a functional programming and OO programming context. Preservation and progress are proved for the core language.

MiniMAO1: An Imperative Core Language for Studying Aspect-Oriented Reasoning by *Curtis Clifton and Gary T. Leavens*

A short and preliminary version of this work has been presented at FOAL 2005. The article presents results of Curtis Clifton's PhD thesis, which was completed in 2005, and which has been supervised by the Gary T. Leavens. The article provides another core AOP language, MiniMAO1. The emphasis is on faithful modeling of interacting features in today's imperative and OO-based AOP languages. In particular, the article covers the subtle issue of around advice that changes the target object before proceeding. The operational semantics and the type system of MiniMAO1 is defined including a proof of soundness. The paper carefully describes and exercises all the necessary meta-theory for such an semantics-based endeavor.

Semantics and Scoping of Aspects in Higher-Order Languages by *Christopher Dutchyn, David B. Tucker and Shriram Krishnamurthi*

This article is a properly extended version of the authors' AOSD 2003 paper ("Pointcuts and advice in higher-order languages"). The article motivates and addresses challenges and benefits of aspects in higher-order languages such as Scheme. An important consideration is the scope of aspects, where static scope admits altering the behavior of join points that are lexically visible within the advice body, and dynamic scope admits altering the behavior of join points that can only be characterized within the dynamic control flow of the advice body. The operational semantics of aspects is defined in terms of an abstract machine; the implementation of a corresponding language extension for Scheme is described.

Typed Parametric Polymorphism for Aspects by *Radha Jagadeesan, Alan Jeffrey and James Riely*

This article continues the authors' research program on the semantics of AOP (cf. ECOOP 2003, CONCUR 2004). The present article studies the incorporation of generic types in OO-based aspect languages. Two models of generics are considered: type-erasure semantics as in Java 1.5 and type-carrying semantics as in C# 2.0. A source-level type system for type-safe aspects+generics is defined. To this end, Featherweight Generic Java is used (while neglecting the orthogonal issue of casting). Preservation and progress are proved. The study and the contribution of this work is timely in so far that the authors manage to show type-unsafety for existing AOP implementations (based on Java 1.5).

Aspect-Oriented Programming for Reactive Systems: Larissa, a Proposal in the Synchronous Framework by *Karine Altisen, Florence Maraninchi and David Stauch*

An early version of this work has been presented at FOAL 2004. This article studies crosscutting concerns for reactive systems. In such systems, crosscutting concerns program structure based on parallel composition. Rather than defining aspects for a specific language for reactive systems (such as Esterel), the authors define a simple set of operators that capture the large class of synchronous languages with their clean and simple definition of parallelism. In this framework, (some) aspects have a suggestive representation based on so-called observers that rely on synchronous broadcasts. However, there are still crosscutting concerns in reactive systems that call for designated AOP language support, e.g., reset and reinitialization. The paper identifies such problems, proposes an aspect language, and proves that aspect weaving preserves the usual behavioral equivalence for reactive systems.

3 Special issue — statistics

This special issue received 10 submissions. The 5 selected papers represent the contributions that are best aligned with the focus of this special issue, while they also adhere to the required, high standards for an archival publication. There were 31 referees involved in the two rounds of reviewing. The original deadline for submission was 1 March, 2005, and all final versions of accepted papers were in the possession of the guest editor about 1 year later. In this process, all papers have been revised and re-reviewed.

4 Acknowledgments

Our gratitude is due to all submitting authors, in general, and to the authors of accepted submissions, in particular, for their willingness to meet all requests for revisions. The quality of the review process was ensured by a distinguished review committee; the referees were selected on a per-paper basis. The names and affiliations of most referees are listed below. Finally, our gratitude goes to Bas van Vlijmen (Editorial Assistant of the SCP Journal) who helped us with the logistics of this special issue.

5 List of referees for the special issue

Paulo Borba	(Universidade Federal de Pernambuco)
John Boyland	(University of Wisconsin)
Daniel Dantas	(Princeton University)
Oege De Moor	(Oxford University)
Kris De Volder	(University of British Columbia)
Rémi Douence	(Ecole des Mines de Nantes)
Erik Ernst	(University of Aarhus)
Robby Findler	(University of Chicago)
Samuel Kamin	(University of Illinois at Urbana-Champaign)
Mika Katara	(Tampere University of Technology)
Shmuel Katz	(Technion)
Andrew Kennedy	(Microsoft Research)
Günter Kniesel	(University of Bonn)
Reino Kurki-Suonio	(Tampere University of Technology)
Luigi Liquori	(Inria)
Marius Marin	(Delft University of Technology)
Hidehiko Masuhara	(University of Tokyo)
Todd Millstein	(UCLA)
Klaus Ostermann	(Technische Universität Darmstadt)
Jeffrey Palm	(Northeastern University)
Henny Sipma	(Stanford University)
David Schmidt	(Kansas State University)
Thomas Schwentick	(Universität Dortmund)
Maximilian Störzer	(Universität Passau)
Mario Südholt	(Ecole des Mines de Nantes)
Kevin Sullivan	(University of Virginia)
Mitchell Wand	(Northeastern University)
Jianjun Zhao	(Shanghai Jiao Tong University)