PROCEEDINGS OF THE

ASPECT-ORIENTED PROGRAMMING WORKSHOP

AT

ECOOP '97

Organizers:
Cristina Lopes, Kim Mens, Bedir Tekinerdogan, and Gregor Kiczales

Includes the workshop report and the position papers.

The workshop report was published in *Workshop Reader of the European Conference on Object-Oriented Programming* (ECOOP), Finland. Springer-Verlag LNCS 1357. June 1997.

Aspect-Oriented Programming Workshop Report

Kim Mens[1], Cristina Lopes[2], Bedir Tekinerdogan[3], and Gregor Kiczales[2]

[1] Vrije Universiteit Brussel,
Department of Computer Science, Programming Technology Lab,
Pleinlaan 2, B-1050 Brussel, Belgium
[2] Xerox PARC, Systems and Practices Laboratory,
3333 Coyote Hill Rd, Palo Alto, CA 94304, USA
[3] University of Twente,
Department of Computer Science, Software Engineering,
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. Whereas it is generally acknowledged that code tangling reduces the quality of software and that aspect oriented programming (AOP) is a means of addressing this problem, there is as yet no clear definition or characterisation of AOP. Therefore, the main goal of the ECOOP'97 AOP workshop was to identify the "good questions" for exploring the idea of AOP.

1 Introduction

Mechanisms for defining and composing abstractions are essential elements of programming languages. They allow programs to be composed up from smaller units, and they support design styles that proceed by decomposing a system into smaller and smaller sub-systems.

The abstraction mechanisms of most current programming languages – sub-routines, procedures, functions, objects, classes, modules and API's – can all be thought of as fitting into a generalised procedure call model. The design style they support is one of breaking a system down into parameterised components that can be called upon to perform some function.

But many systems have properties that do not necessarily align with the system's functional components. Failure handling, persistence, communication, replication, coordination, memory management, real-time constraints and many others are aspects of a system's behaviour that tend to cut-across groups of functional components. While these aspects can be thought about and analysed relatively separately from the basic functionality, programming them using current component-oriented languages tends to result in these aspects being spread throughout the code. The source code becomes a tangled mess of instructions for different purposes.

This "tangling" phenomenon is at the heart of much needless complexity in existing software systems. It increases the dependencies between the functional

components. It distracts from what the components are supposed to do. It introduces numerous opportunities for programming errors. It makes the functional components less reusable. In short, it makes the source code difficult to develop, understand and evolve.

A number of researchers [KLM+97] have begun working on approaches to this problem that allow programmers to express each of a system's aspects of concern in a separate and natural form, and then automatically combine those separate descriptions into a final executable form using automatic tools. These approaches have been called aspect-oriented programming (AOP).

In this workshop, rather than focussing on the idea of automatic weaver tools, a more general notion of AOP was adopted: AOP was regarded as a general concept or mechanism to solve the problem of modelling the different aspects of concern in a system. The purpose of the workshop was to bring together researchers and practitioners working in the area of AOP or related areas to discuss the current status of AOP research.

2 About the Workshop

The second workshop on *aspect-oriented programming* was organised by Cristina Videira Lopes, Gregor Kiczales, Kim Mens and Bedir Tekinerdogan on June 10 during the 11th European Conference on Object-Oriented Programming in Jyväskylä, Finland. (The first AOP workshop — the "AOP friends meeting" — was held at Xerox PARC in conjunction with OOPSLA'96.)

All participants were encouraged to submit a short position paper and the workshop was organised around the common tendencies detected in these position papers, such as:

1. What exactly are *aspects*? How can they be identified or characterised? [Meu97,MJV+97]
2. What is the difference between an aspect and a *component*? How do components and aspects interact? [HOT97,Lam97,Van97]
3. How to *weave*? (I.e. how to merge the base component program and the different aspect programs into a final executable form.) [Lam97]
4. Need for a theoretical foundation for AOP. [Meu97]
5. How to expand the use of asperts to other phases of the software development life-cycle: requirements, analysis, architecture, design, implementation, maintenance, ... [Aks97,HOT97,MJV+97,Mul97,Wer97]
6. What are the relationships or differences between AOP and other approaches or programming paradigms and especially between AOP, reflection, open implementations and meta-object protocols? (For example, is AOP better than a general framework like reflection?) [CES97,Meu97,DC97,MJV+97,Lam97]
7. Visual representations of AOP. (For example, visual presentation of relationships between components/aspects, graphical representations of aspects and aspect weaving, ...) [HOT97,Van97,Wer97]