

# GEMA: A Generic Model for AOP

Katharina Mehner<sup>\*†</sup>, Awais Rashid

Computing Department, Lancaster University, Lancaster LA1 4YR, UK

[mehner@upb.de](mailto:mehner@upb.de), [awais@comp.lancs.ac.uk](mailto:awais@comp.lancs.ac.uk)

## Abstract

A common understanding of what is AOP is still missing and attempts are rare in literature. However, it is needed to determine whether a suggested approach is AOP. At the same time it can help in building many kinds of tools from weavers and interpreters to debuggers and analysers by identifying generalities, e.g., in the form of common architectures or data formats, and thereby help in avoiding adhoc-implementations. Our aim is to provide a model of AOP which draws upon established AOP techniques to capture essential features. For instance, it should allow drawing the line between a meta-object protocol and an AOP environment. The model should be generic in the sense that it captures basic yet essential commonalities but can adapt to variability (and also optional features). For instance, it should be instantiatable to approaches like AspectJ and Adaptive Programming. We take AOP literally, i.e., we focus on programming but not on other phases of software development. The intended scope of this work covers linguistic approaches to AOP and non-linguistic approaches, e.g., based on existing programming languages or reflection mechanisms. Most AOP approaches aim at improving separation of concerns with respect to existing programming languages such as OOP, functional or logic programming. This is achieved by capturing crosscutting code in one place, the so called aspect (or parts thereof). At the heart of each AOP approach lies a composition mechanisms for integrating the separated pieces of code. These two features which we denote as structuring and composition are the basic ideas on which every AOP approach builds and are therefore the starting point for our generic AOP model (GEMA). Each feature is classified according to several dimensions. The dimensions of structuring are abstraction, modularisation, encapsulation, information hiding, and hierarchy. The dimensions of composition are joinpoints, dominance (some approaches identify dominant components with which aspects are to be integrated while others treat all components equally), composition rules (including conflict resolution), and composition time. Each dimension can be further classified by facets. For instance, the joinpoints can take various forms. Many AOP techniques use method call interception, e.g., AspectJ's advice weaves, which has been extended to more general runtime event interception lately. This facet is also called functional correspondence. We speak of component correspondence if behaviourally unrelated slices of data and functions are specified separately but are identified to belong, for instance, to the same class. This approach is supported by AspectJ's introduction weaves and by HyperJ. Data correspondence occurs if the same entity is present in several components but has to be identified as one entity. This is present as a particular form of merge in HyperJ. Attaching traversal strategies to class graphs as found in Adaptive Programming can be seen as a more high-level correspondence between compositional entities. In the talk we will present the current state of our generic AOP model and insights into its derivation.

---

\* On the leave of Department of Computer Science, University of Paderborn, D-33095 Paderborn, Germany

† This work is supported by Lancaster University, Research Committee grant FasOP (Foundations of Aspect-Oriented Programming)