Flexible Pointcut Implementation: An Interpreted Approach

Ali Assaf*, Jacques Noyé**
Département informatique, École des Mines de Nantes

*Ali.Assaf@emn.fr,
**Jacques.Noye@emn.fr

Abstract. One of the main elements of an Aspect-Oriented Programming (AOP) language or framework is its *pointcut language*. A *pointcut* is a predicate which selects program execution points and determines at which points the execution should be affected by an aspect. Experimenting with AspectJ shows that two basic primitive pointcuts, `call` and `execution`, dealing with method invocation from the caller and callee standpoints, respectively, lead to confusion. This is due to a subtle interplay between the use of static and dynamic types to select execution points, dynamic lookup, and the expectation to easily select the caller and callee execution points related to the same invocation. As a result, alternative semantics have been proposed but have remained paper design.

In this article, we reconsider these various semantics in a practical way by implementing them using CALI, our *Common Aspect Language Interpreter*. This framework reuses both Java as a base language and AspectJ as a way to select the program execution points of interest. An additional interpretation layer can then be used to prototype interesting AOP variants in a full-blown environment. The paper illustrates the benefits of applying such a setting to the case of the `call` and `execution` pointcuts. We show that alternative semantics can be implemented very easily and exercised in the context of AspectJ without resorting to complex compiler technology.

1 Introduction

AspectJ is the most popular AOP language (Kiczales et al., 2001; Filman et al., 2005). It extends Java with a new type of code structuring entity, called *aspect*, which contains the definition of *pointcuts* and *advices* in addition to standard Java members like fields and methods. A *pointcut* selects execution points of the so-called *base program* where extra code (the *advice*) has to be executed. These points are called *join points*. At the user level, the properties of the pointcut language like expressiveness, obliviousness, clarity of semantics, comprehensibility, flexibility, extensibility, etc. are important issues in the design of AOP languages. At the implementation level, a *weaver* is responsible for *weaving* the selection of the join points into the base program. As soon as static information is available, for instance types, and join-point selection relies on this information, part of the selection process can be dealt with statically (at