Science of Computer Programming 72 (2008) 22–30

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Flexible metaprogramming and AOP in Java

Éric Tanter [a,*], Rodolfo Toledo [a], Guillaume Pothier [a], Jacques Noyé [b]

[a] PLEIAD Lab, Computer Science Department (DCC), University of Chile–Santiago, Chile
[b] OBASCO Project-Team EMN/INRIA, LINA–Nantes, France

ARTICLE INFO

Article history:
Received 30 April 2006
Received in revised form 3 September 2007
Accepted 18 October 2007
Available online 25 April 2008

Keywords:
Metaprogramming
Reflection
Aspect-oriented programming
Reflex
Java

ABSTRACT

Advanced programming techniques such as metaprogramming and computational reflection, as well as the more recent paradigm of aspect-oriented programming (AOP), serve important objectives of software engineering such as modularization and adaptability. In this tool presentation paper, we briefly overview this area and present Reflex, a portable tool for flexible metaprogramming and AOP in Java.

Reflex provides both structural and behavioral facilities adopting a uniform model of partial reflection. This allows selective and fine-grained control of where and when reflection occurs. The facilities of Reflex make it easy to experiment with (combinations of) advanced uses of AOP and reflection without reinventing the wheel or being limited to a specific AOP language.

1. Introduction

Research in programming languages has been driven by the need to achieve well-modularized software respecting the principle of Separation of Concerns [1,2]. Good modularization serves many software engineering properties such as understandability, extensibility, reusability, etc. It also helps to make software more *adaptable* [3,4], since for a given concern to be adaptable (possibly dynamically) it first has to be cleanly modularized. Work on computational reflection [5,6], metaprogramming, and more recently, aspect-oriented programming (AOP) [7,8], has been a fruitful path to achieve better modularization and adaptation in many systems, such as middleware [9], concurrent systems [10,11], distributed programming [12–15], operating systems [16,17], user interfaces [18], context-aware applications [19,20], etc.

The Java programming language only offers a limited set of reflective abilities, which have been progressively extended as the language matured. Still, many fundamental reflective features are missing. This is why many reflective and/or aspect-oriented extensions of Java have been proposed; just to name a few: Javassist [21], for structural reflection at load time, Kava [22], for runtime behavioral reflection, and on the AOP side, AspectJ [23], the most popular Java language extension for AOP, and frameworks such as AspectWerkz [24], JAC [25], and JAsCo [26].

This paper gives an overview of Reflex, a portable Java framework for flexible metaprogramming and AOP. Reflex bridges the gap between metaprogramming and reflection on one side and AOP on the other side, and hence provides advanced users with a versatile kernel for experimenting with AOP concepts and language features [27–30]. It has been applied in concrete application domains such as concurrent systems [11], distributed systems [15,31], and context-aware applications [20]. It is an open source project distributed under the MIT license, and the Reflex website[1] gives access to many resources, such as documentation and a tutorial, a subversion repository, mailing lists, and publications.

* Corresponding author.
E-mail addresses: etanter@dcc.uchile.cl (É. Tanter), rtoledo@dcc.uchile.cl (R. Toledo), gpothier@dcc.uchile.cl (G. Pothier), noye@emn.fr (J. Noyé).

[1] http://pleiad.dcc.uchile.cl/reflex

0167-6423/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2007.10.005

In the next section, we give a bit more background information on metaprogramming, reflection and AOP. We then present the key features of Reflex in Section 3, by exposing its underlying model and concepts. API details are not addressed here, but the reader is referred to the Reflex tutorial on the website. Section 4 addresses the use of Reflex in practice. We finally briefly discuss related systems in Section 5. Section 6 concludes with on-going and future work.

2. Metaprogramming and AOP

After the seminal work of Brian C. Smith on computational reflection [5,32], and the marriage of reflection and object-oriented programming by Pattie Maes [6], many attempts have been made to apply so-called *metaobject protocols* (MOPs) [33] for achieving separation of concerns [34]. The basic idea is that the semantics of a base program is modularly extended or modified by appropriate *metaobjects*. A metaobject is given control over *reifications* of the structure or behavior of the underlying program, *i.e.* objects describing otherwise implicit elements of a program. Hence metaobjects can take care of particular *concerns* of the application, such as authentication or invariant checking, while the base application is mostly unaware of these concerns.

This led to the issue of *metalevel engineering* [35], that is, the organization of metalevel entities in ways that are satisfactory with respect to the traditional engineering principles of composability, extensibility, and flexible granularity.[2] These issues have given rise to many reflective architectures, exploring different approaches to metalevel engineering. A particularly interesting one is the *operational* decomposition proposed by McAffer [35]. McAffer distinguishes between two approaches to reflection, which consist of either starting from the base-level language structural elements (*e.g.* classes), or from the basic *operations* defining the computational behavior of an object (message send and receive, field access, object creation, etc.). He refers to these approaches as the top-down and the bottom-up approach, respectively. One could alternatively refer to them as a structural and a behavioral approach. McAffer justifies the use of the second approach as it is more flexible in terms of granularity and makes it possible to describe a wider range of behavior models.

At the same time, work on open implementations [18] was facing the same issues of metalevel locality of change and engineering. Granularity of metalevel entities directly affects locality of change: altering the definition of a metaclass affects all the instances of that class, while changing the metalevel entity representing one single method invocation leaves the rest of the program intact. Furthermore, Kiczales noticed that sometimes the metalevel concepts that are most natural to use actually *crosscut* the concepts at the base level [36]. This led his group to focus on this crosscutting issue and to eventually come up with the paradigm of Aspect-Oriented Programming [7] (AOP). AOP is now a very active research area [8].[3]

AOP puts forward a new kind of module called an *aspect*, which is the modular definition of a crosscutting concern. An aspect can act on a program by synchronizing with it at *join points*, usually defined as program execution points where an aspect applies, and performing its action, often called an *advice*, a term inherited from Lisp. Although the most famous join point model is the dynamic join point model, whereby a join point is simply a program execution point, which greatly resembles the operational decomposition of McAffer, a join point model can also refer to other program properties (*e.g.* data flow graphs [7], traces [37], structural properties [38]). Individual join points are grouped together by means of *pointcuts*, which can be seen as queries on the program structure and the program execution.

3. Reflex

Reflex is a portable library that extends Java with structural and behavioral reflective facilities. We first describe the uniform model of partial reflection that lies at the heart of Reflex, before surveying the structural and behavioral facilities of Reflex. We end this section with a brief discussion of Reflex as a versatile kernel for AOP.

3.1. Uniform model of partial reflection

Partial reflection consists in providing reflective features only where and when needed, in the most selective manner, in order to reduce the overhead associated with full reflection [27]. The underlying model of partial reflection of Reflex is that of explicit *links* binding a *cut* to an *action*. A cut specifies which program elements are of interest, the action specifies what to do on these program elements. The link is an *explicit* entity binding both, characterized by several attributes. Links are the basic unit of specification in Reflex, and can be defined either eagerly before an application starts, or dynamically while the application is running.

The cut of a link is defined via *selection predicates*, as illustrated later in this section, and the action is implemented in a *metaobject*. A metaobject can be *any* standard Java object, provided it implements the expected protocol. There are two kinds of links: *structural links* and *behavioral links*. The former are used to perform structural reflection at load time, while the latter are used to perform behavioral reflection at run time.

---

[2] Granularity here refers to the scope of metalevel entities: a coarse-grained metalevel decomposition can for instance provide one metaobject representing the expression interpreter, while a finer-grained decomposition can provide a metalevel entity per class (metaclasses), or per object, or even finer, per execution step.

[3] See the community site at http://aosd.net/.