Conference Paper

Kava - A Reflective Java Based on Bytecode Rewriting.

Ian Welch Robert J. Stroud

DOI: 10.1007/3-540-45046-7_9 Conference: Reflection and Software Engineering, Papers from OORaSE 1999, 1st OOPSLA Workshop on Reflection and Software Engineering, Denver, CO, USA, November 1999
Source: DBLP

ABSTRACT Current implementations of reective Java typically either require access to source code, or require a modied Java platform. This makes them unsuitable for applying reection to Commercial-o-the - Shelf (COTS) systems. T [more]

Full-text

Download full-text

Available from: Ian Welch, Sep 24, 2015

Kava - A Reflective Java based on Bytecode Rewriting

Ian Welch and Robert J. Stroud

d Kingdom NE1 7RU
cl.ac.uk,
l.ac.uk/people/
ud}

tive Java typically either
dified Java platform. This
n to Commercial-off-the-

Shelf (COTS) systems. The high level nature of Java bytecode makes on-the-fly rewritings of class files feasible and this has been exploited by a number of authors. However, in practice working at bytecode level is error prone and leads to fragile code. We propose using metaobject protocols in order to specify behavioural changes and use standard bytecode rewritings to implement the changes. We have developed a reflective Java called *Kava* that provides behavioural runtime reflection through the use of bytecode rewriting of Java classes. In this paper we discuss the binary rewriting approach, provide an overview of the *Kava* system and provide an example of an application of *Kava*.

1 Introduction

We are interested in the problems of applying non-functional requirements to Commercial Off-the-Shelf (COTS) software components. In an environment such

REFERENCES (25) **CITED IN** (41)

Source

"It is easy to see how our proposed architecture could be implemented for software platforms based on the Java language runtime. Java-based systems already support both code manifests and strong cryptographic code identity [15]; many already contact the platform creators through an online service for sending error reports and retrieving updates; and it has been demonstrated that it is simple to accommodate both the identification and hooking required for the mediation framework [13] [54]. Indeed, architectures similar to our proposal have already been implemented for Java-based application server platforms, the Wily Interscope management and monitoring system being one example [55]. "

Article: Ad hoc extensibility and access control.

Úlfar Erlingsson · John MacCormick

[Show abstract]

ACM SIGOPS Operating Systems Review 07/2006; 40:93-101. DOI:10.1145/1151374.1151393

Source

"Moving reflective mechanisms from the programming to the system level is a relatively recent trend [3] [7]. Most of the existing reflective languages support reflection through the introduction of linguistic hooks 3 . "

Conference Paper: Implementing the essence of reflection: a reflective run-time environment.

Massimo Ancona · Walter Cazzola

[Show abstract]

Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004; 01/2004

Source

"The former approach is used for Java extensions such as Kava [17] and Javassist [2]. The latter is adopted by languages such as OpenC++ [1] and OpenJava [15] "

Conference Paper: Introducing distribution into applications: a reflective approach for transparency and dynamic fine-grained object allocation

A. Di Stefano · G. Pappalardo · E. Tramontana

[Show abstract]

Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on; 02/2002

Show more

Note: This list is based on the publications in our database and might not be exhaustive.

SIMILAR PUBLICATIONS

Side effect monitoring for Java using bytecode rewriting
Manuel Geffken, Peter Thiemann

Mobile Process Resumption In Java Without Bytecode Rewriting
Matthew Sowders, Jan Baekgaard Pedersen

Optimistic-parallel process-oriented DES in Java using Bytecode Rewriting
Andreas Kunert

Kava – A Reflective Java Based on Bytecode Rewriting.

CONFERENCE PAPER · JANUARY 1999

DOI: 10.1007/3-540-45046-7_9 · Source: DBLP

2 AUTHORS, INCLUDING:

Ian Welch
Victoria University of Wellington
91 PUBLICATIONS **730** CITATIONS

Kava - A Reflective Java based on Bytecode Rewriting

Ian Welch and Robert J. Stroud

University of Newcastle-upon-Tyne, United Kingdom NE1 7RU
{I.S.Welch, R.J.Stroud}@ncl.ac.uk,
WWW home page:http://www.cs.ncl.ac.uk/people/
{I.S.Welch, R.J.Stroud}

Abstract. Current implementations of reflective Java typically either require access to source code, or require a modified Java platform. This makes them unsuitable for applying reflection to Commercial-off-the-Shelf (COTS) systems. The high level nature of Java bytecode makes on-the-fly rewritings of class files feasible and this has been exploited by a number of authors. However, in practice working at bytecode level is error prone and leads to fragile code. We propose using metaobject protocols in order to specify behavioural changes and use standard bytecode rewritings to implement the changes. We have developed a reflective Java called *Kava* that provides behavioural runtime reflection through the use of bytecode rewriting of Java classes. In this paper we discuss the binary rewriting approach, provide an overview of the *Kava* system and provide an example of an application of *Kava*.

1 Introduction

We are interested in the problems of applying non-functional requirements to Commercial Off-the-Shelf (COTS) software components. In an environment such as Java, components are usually supplied in a compiled form without source code, and can be integrated into a system at runtime.

Metaobject protocols [12] offer a principled way of extending the behaviour of these components. Metaobjects can encapsulate the behavioural adaptations necessary to satisfy desirable non-functional requirements (NFRs) such as fault tolerance or application level security [1][2][18][19] transparently at the metalevel. Ideally we want to apply these metaobjects to compiled code that executes on a standard Java platform.

The Java 2 Reflection package `java.lang.reflect` provides introspection, dynamic dispatch and the ability to generate proxies for classes on-the-fly. However, this is not sufficient to build a rich metaobject protocol that can be applied transparently. There are a number of alternative extensions for Java that provide more powerful and more transparent reflection. However, they all have flaws that do not make them applicable to the problem of adapting components. These flaws include the requirement for customised Java Virtual Machines (JVMs), limited reflective capabilities, or weak non-bypassability. The term non-bypassability

refers to the binding between the base level and the meta level. For a number of NFRs such as security the meta level should never be able to be bypassed. This is what we term strong non-bypassability. However, in a number of implementations the techniques used to implement the bindings are easily bypassed. We refer to this as weak non-bypassability.

We have produced our own implementation of a reflective extension for Java called *Kava* that provides a rich metaobject protocol, requires only a standard JVM and provides strong non-bypassability. *Kava* implements a runtime behavioural metaobject protocol through the application of standard byte code rewritings, and behavioural adaptation is implemented using Java metaobject classes. This is to be distinguished from structural reflection where a metaobject protocol provides an interface for a programmer who wants to change the actual structure of an object.

The rest of the paper is organized as follows. In section two we provide a review of different approaches to implementing reflection in Java. Section three introduces the *Kava* metaobject protocol. Section four explains the byte code rewriting approach. Section five gives an example of an application of *Kava*. Finally section six provides some conclusions about the general approach.

A prototype implementation of *Kava* has been completed and is available from *http* : *//www.cs.ncl.ac.uk/people/i.s.welch/kava*.

2 Review of Reflective Java Implementations

In this section we briefly review a number of reflective Java implementations and attempt to categorize them according to the point in the Java class lifecycle that reflection is implemented.

The Java class lifecycle is as follows. A Java class starts as source code that is compiled into byte code, it is then loaded by a class loader into the JVM for execution, where the byte code is further compiled by a Just-In-Time compiler into platform specific machine code for efficient execution.

Different reflective Java implementations introduce reflection at different points in the lifecycle. The point at which they introduce reflection characterizes the scope of their capabilities. In order to realise reflective control by a metalevel the baselevel system is modified through the addition of traps on operations. These traps are known as metalevel interceptions [26]. For example, in *Reflective Java* method calls sent to the base object are brought under control of an associated meta object by trapping each method call to the base object. These traps are added at the source code stage of the lifecycle making it necessary to have access to the source code. In contrast *MetaXa* adds the traps into the JVM, here the implementation of the dispatch mechanism of the JVM is changed in order to take control over method calls. As the traps are added to the runtime system source code is no longer required. The drawback of this approach is that unlike *Reflective Java* a specialized JVM must be used.

Table 1 summarizes the features of the different reflective Java implementations.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.