Information and Software Technology 56 (2014) 199–219

Contents lists available at ScienceDirect

Information and Software Technology

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

A hybrid class- and prototype-based object model to support language-neutral structural intercession

Francisco Ortin [a,*], Miguel A. Labrador [b], Jose M. Redondo [a]

[a] University of Oviedo, Computer Science Department, Calvo Sotelo s/n, 33007 Oviedo, Spain
[b] University of South Florida, Department of Computer Science and Engineering, 4202 East Fowler Avenue, ENB118 Tampa, FL, USA

ARTICLE INFO

Article history:
Received 25 July 2012
Received in revised form 4 September 2013
Accepted 6 September 2013
Available online 17 September 2013

Keywords:
Structural intercession
Duck typing
Prototype-based object model
Reflection
Virtual machine
Dynamic languages

ABSTRACT

Context: Dynamic languages have turned out to be suitable for developing specific applications where runtime adaptability is an important issue. Although .NET and Java platforms have gradually incorporated features to improve their support of dynamic languages, they do not provide intercession for every object or class. This limitation is mainly caused by the rigid class-based object model these platforms implement, in contrast to the flexible prototype-based model used by most dynamic languages.
Objective: Our approach is to provide intercession for any object or class by defining a hybrid class- and prototype-based object model that efficiently incorporates structural intercession into the object model implemented by the widespread .NET and Java platforms.
Method: In a previous work, we developed and evaluated an extension of a shared-source implementation of the .NET platform. In this work, we define the formal semantics of the proposed reflective model, and modify the existing implementation to include the hybrid model. Finally, we assess its runtime performance and memory consumption, comparing it to existing approaches.
Results: Our platform shows a competitive runtime performance compared to 9 widespread systems. On average, it performs 73% and 61% better than the second fastest system for short- and long-running applications, respectively. Besides, it is the JIT-compiler approach that consumes less average memory. The proposed approach of including a hybrid object-model into the virtual machine involves a 444% performance improvement (and 65% less memory consumption) compared to the existing alternative of creating an extra software layer (the DLR). When none of the new features are used, our platform requires 12% more execution time and 13% more memory than the original .NET implementation.
Conclusion: Our proposed hybrid class- and prototype-based object model supports structural intercession for any object or class. It can be included in existing JIT-compiler class-based platforms to support common dynamic languages, providing competitive runtime performance and low memory consumption.

1. Introduction

Dynamic languages have recently turned out to be suitable for specific scenarios such as Web development, rapid prototyping, developing systems that interact with data that change unpredictably, dynamic aspect-oriented programming, and any kind of runtime adaptable or adaptive software. The main benefit of these languages is the simplicity they offer to model the dynamicity that is sometimes required to build high context-dependent software. Common features of dynamic languages are meta-programming, reflection, mobility, and dynamic reconfiguration and distribution.

Taking Web engineering as an example, Ruby [1] has been successfully used together with the Ruby on Rails framework to create database-backed Web applications [2]. This framework has confirmed the simplicity of implementing the DRY (Do not Repeat Yourself) [3] and the Convention over Configuration [2] principles with this kind of languages. Nowadays, JavaScript [4] is being widely employed to create interactive Web applications with AJAX (Asynchronous JavaScript And XML) [5], while PHP (PHP Hypertext Preprocessor) is one of the most popular languages for developing Web-based views. Python [6] is used for many different purposes; two well-known examples are the Zope application server [7] (a framework for building content management systems, intranets and custom applications) and the Django Web application framework.

Due to the recent success of dynamic languages, statically typed languages – such as Java and .NET – are gradually incorporating more dynamic features into their platforms. Taking Java as an

* Corresponding author. Tel.: +34 985 10 3172.
E-mail addresses: ortin@lsi.uniovi.es (F. Ortin), labrador@cse.usf.edu (M.A. Labrador), redondojose@uniovi.es (J.M. Redondo).
URLs: http://www.di.uniovi.es/~ortin (F. Ortin), http://www.csee.usf.edu/~labrador/ (M.A. Labrador).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.09.002

example, the reflection API became part of the Java platform with its release 1.1. This API offers introspection services to examine the structures of object and classes at runtime, plus object creation and method invocation – involving a substantial performance overhead. The dynamic proxy class API was added to Java 1.3. It allows defining a class at runtime that implements any interface, funneling all its method calls to an `InvocationHandler`. The Java `instrument` package (included in Java SE 1.5) provides services that allow Java agents to instrument programs running on the JVM. This package has been used to implement JAsCo, a fast dynamic AOP platform [8]. Together with other tools such as BCEL [9] and Javassist [10], these agents have also been successfully used in the implementation of application servers such as Spring Java and JBoss, obtaining good runtime performance. The Java Scripting API added to Java 1.6 permits dynamic scripting programs to be executed from, and have access to, the Java platform [11]. Finally, the Java Specification Request 292 [12] has been incorporated to the Java 1.7 Standard Edition. It adds the new `invokedynamic` opcode to the Java Virtual Machine (JVM) and the `java.lang.invoke` package to the platform [12], making it easier to implement dynamically typed languages in the Java virtual machine. Its main advantage is a user-defined linkage mechanism to postpone method call-sites resolution until runtime.

This trend has also been observed in the .NET platform. The Dynamic Language Runtime (DLR) has been included as part of the .NET framework 4.0 [13]. The DLR adds to the .NET platform a new layer that provides services to facilitate the implementation of dynamic languages over the platform [14]. Moreover, Microsoft has included the new `dynamic` type to C# 4.0, allowing the programmer to write dynamically typed code in a statically typed programming language. With this new characteristic, C# 4.0 offers direct access to code in IronPython, IronRuby and the JavaScript code in Silverlight, making use of the DLR services.

The DLR is built over the .NET virtual machine (the CLR, Common Language Runtime) to provide dynamic typing features over a statically-typed and class-based platform. It offers the common primitives provided by dynamic languages, such as structural intercession and duck typing, that the CLR does not support; it also simulates the prototype-based object model implemented by many reflective dynamic languages [15]. This extra layer over the virtual machine involves some drawbacks. The first one is that the specific features of dynamic languages are not provided for every object or class in the system. A C# programmer can only change the structure of `ExpandoObject` instances (see Section 2.3). Another limitation is that the introspective services of the platform do not take effect with `ExpandoObject`s. An additional major disadvantage is that this new software layer commonly involves a runtime performance penalty (see Section 4).

In order to overcome these limitations, we propose the addition of the specific features provided by most dynamic languages (such runtime structural intercession and duck typing) at the virtual machine level, supplying these services for any object or class. Therefore, the benefits of widely-used statically-typed platforms such as Java and .NET will be complemented with the runtime adaptiveness of dynamic languages. The use of the very same virtual machine for both types of languages will also facilitate the future interoperation between them.

The main contribution of this paper is the definition of a hybrid class- and prototype-based model valid to support structural intercession and duck typing at the virtual machine level of class-based platforms such as Java and .NET, offering these new features for any object or class. In a previous work, we extended a shared source implementation of .NET (the SSCLI, also known as Rotor) to provide some of the introspection services provided by most existing reflective languages [16]. In this work, we have extended that implementation to include the proposed hybrid class- and prototype-based model, validated with a lightweight formalization tool. Runtime performance and memory consumption of that implementation has been evaluated and compared with other existing approaches. We refer to our platform as Reflective Rotor or ЯRotor.

The rest of the paper is structured as follows. Section 2 presents the existing approaches to provide structural intercession, and our proposed model is formalized in Section 3. Section 4 presents an evaluation of runtime performance and memory consumption. Section 5 discusses the related work, and Section 6 concludes and presents future work.

2. Existing approaches to provide structural intercession

There are two main approaches to provide structural intercession: the class-based and the prototype-based object models. The .NET platform implements a hybrid approach, but its reflective services are not offered for every object or class (Section 2.3). As we will see, this limitation is mainly caused by the rigid class-based object model this platform actually implements in contrast to the flexible prototype-based model used by most dynamic languages. This section first defines the basis of reflection; afterwards, the existing approaches to provide structural intercession are analyzed.

Reflection is the capability of a computational system to reason about and act upon itself, adjusting itself to changing conditions [17]. In a reflective language, the computational domain is enhanced with its self-representation, offering its structure and semantics as computable data at runtime. Reflection has been recognized as a suitable tool to aid the dynamic evolution of running applications, being the primary technique to obtain meta-programming, adaptiveness, and dynamic reconfiguration features of dynamic languages [18]. Computational reflection is the activity performed by a computational system when doing computation about (and by possibly affecting) its own computation [17].

Introspection is the reflection level that allows the inspection, but not the modification, of the program self-representation. Both Java and .NET platforms offer introspection by means of the `java.lang.reflect` package and the `System.Reflection` namespace, respectively. With these services, the programmer can obtain information about classes, objects, methods and fields at runtime. On the other hand, *intercession* is the ability of a program to modify its own execution state, including the customization of its own interpretation or meaning. Adding or removing object fields at runtime is a typical example of intercession.

Both introspection and intercession are classified as *structural* reflection when the system structure is the information reflected (offered to the programmer as data). In case the program structure is modified (i.e., structural intercession), changes will be reflected at runtime. An example of this kind of reflection is inspecting (or modifying) the structure of a class by the program itself. *Behavioral* reflection is concerned with the ability to access to the system semantics. For instance, MetaXa (formerly called MetaJava [19]) is a Java extension that offers the programmer the dynamic modification of the method dispatching mechanism.

2.1. Structural intercession in class-based languages

SmallTalk and CLOS are two examples of class-based languages that provide structural intercession. Their class-based model does not provide a consistent support for every structural intercession primitive. This fact was first noticed and partially solved in the field of object-oriented database management systems [20]. In this area, objects are stored but their structure, and even their types (classes), can be altered afterwards as a result of software evolution.