Journal of Visual Languages and Computing 25 (2014) 346–362

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

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

Static type information to improve the IDE features of hybrid dynamically and statically typed languages ☆

Francisco Ortin [a,*], Francisco Moreno [b], Anton Morant [c]

[a] University of Oviedo, Computer Science Department, C/Calvo Sotelo s/n, 33007 Oviedo, Spain
[b] Alisys Software S.L.U., C/Menendez Valdes 40, 33201 Gijon, Spain
[c] University of Oxford, Wolfson College, Linton Road, OX26UD Oxford, UK

ARTICLE INFO

Article history:
Received 26 October 2012
Received in revised form
7 April 2014
Accepted 21 April 2014
Available online 28 April 2014

Keywords:
Hybrid dynamic and static typing
IDE support
Type inference
Code completion
Separation of concerns
Plug-in
Visual Studio

ABSTRACT

The flexibility offered by dynamically typed programming languages has been appropriately used to develop specific scenarios where dynamic adaptability is an important issue. This has made some existing statically typed languages gradually incorporate more dynamic features to their implementations. As a result, there are some programming languages considered hybrid dynamically and statically typed. However, these languages do not perform static type inference on a dynamically typed code, lacking those common features provided when a statically typed code is used. This lack is also present in the corresponding IDEs that, when a dynamically typed code is used, do not provide the services offered for static typing. We have customized an IDE for a hybrid language that statically infers type information of dynamically typed code. By using this type information, we show how the IDE can provide a set of appealing services that the existing approaches do not support, such as compile-time type error detection, code completion, transition from dynamically to statically typed code (and vice versa), and significant runtime performance optimizations. We have evaluated the programmer's performance improvement obtained with our IDE, and compared it with similar approaches.

1. Introduction

Dynamic languages have turned out to be suitable for specific scenarios such as rapid prototyping, Web development, interactive programming, 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.

Taking the Web engineering area as an example, Ruby [1] has been successfully used together with the Ruby on Rails framework for creating database-backed Web applications [2]. This framework has confirmed the simplicity of implementing the DRY (*Don't Repeat Yourself*) [3] and the *Convention over Configuration* [2] principle with this kind of languages. Nowadays, JavaScript [4] is being widely employed to create interactive Web applications with AJAX [5], while PHP is one of the most popular languages to develop Web-based views. Python [6] is used for many different purposes, the Zope application server [7] and the Django Web application framework [8] being two well-known examples.

☆ This paper has been recommended for acceptance by Shi Kho Chang.

* Corresponding author.
E-mail addresses: ortin@lsi.uniovi.es (F. Ortin), francisco.moreno@alisys.net (F. Moreno), anton.morant@comlab.ox.ac.uk (A. Morant).
URL: http://www.di.uniovi.es/~ortin (F. Ortin).

http://dx.doi.org/10.1016/j.jvlc.2014.04.002
1045-926X/

Due to the recent success of dynamic languages, other statically typed ones such as Java and C# are gradually incorporating more dynamic features into their platforms. Taking C# as an example, the .Net platform was initially released with introspective and low-level dynamic code generation services. Version 2.0 included dynamic methods and the `CodeDom` namespace to generate the structure of high-level source code documents. The Dynamic Language Runtime (DLR) adds to the .Net platform a set of services to facilitate the implementation of dynamic languages. A new `dynamic` type has been included in C# 4.0 to support the dynamically typed code. When a reference is declared as `dynamic`, the compiler performs no static type checking, postponing all the type verifications until runtime [9]. With this new characteristic, C# 4.0 offers direct access to the dynamically typed code in IronPython, IronRuby and the JavaScript code in Silverlight.

Java also seems to follow this trend. The last addition to support features commonly provided by dynamic languages has been the Java Specification Request (JSR) 292, partially included in Java 7. The JSR 292 incorporates the new `invokedynamic` opcode to the Java Virtual Machine so that it can run dynamic languages with a performance level comparable to that of Java itself [10].

The flexibility of dynamic languages is, however, counteracted by limitations derived from the lack of static type checking. This deficiency implies two major drawbacks: no early detection of type errors and less opportunities for compiler optimizations. Static typing offers the programmer the detection of type errors at compile time, making it possible to fix them immediately rather than discovering them at runtime—when the programmer's efforts might be aimed at some other task, or even after the program has been deployed. Moreover, since runtime adaptability of dynamic languages is mostly implemented with dynamic type systems, runtime type inspection and checking commonly involve a significant performance penalty [11].

Since both static and dynamic typing approximations offer different benefits, there have been former works to provide both typing approaches in the same language (see Section 5). Meijer and Drayton [12] maintain that instead of providing programmers with a black or white choice between static and dynamic typing, it could be useful to strive for softer type systems. Static typing allows earlier detection of programming mistakes, better documentation, more opportunities for compiler optimizations, and increased runtime performance. Dynamic typing languages provide a solution to a kind of computational incompleteness inherent to statically typed languages, offering, for example, storage of persistent data, inter-process communication, dynamic program behavior customization or generative programming [12]. Therefore, there are situations in programming when one would like to use dynamic types even in the presence of advanced static type systems [13]. That is, *static typing where possible, dynamic typing when needed* [12].

As proposed by Meijer and Drayton, we break the programmers' black or white choice between static and dynamic typing. We have developed a programming language called *StaDyn* that provides both type systems [14]. *StaDyn* is an extension of C# 3.0, which supports static and dynamic typing. *StaDyn* permits the straightforward development of adaptable software and rapid prototyping, without sacrificing application robustness and runtime performance. The programmer indicates whether high flexibility is required (dynamic typing) or stronger type checking (static) is preferred. It is also possible to combine both approaches, making parts of an application more flexible, whereas the rest of the program maintains its robustness and runtime performance.

The main contribution of this paper is a visual IDE that takes advantage of the specific features of hybrid statically and dynamically typed languages, showing how the type information gathered by the compiler can be used to provide new features plus others that are commonly offered for statically typed code only. The *StaDyn* IDE separates the *dynamism* concern [15] facilitating the transition from rapidly developed prototypes to final robust and efficient applications, detects many type errors of dynamically typed code at compile time, provides code completion for dynamically typed code, and performs significant code optimizations.

The rest of this paper is structured as follows. In Section 2, the *StaDyn* IDE is described, emphasizing the new features added to Visual Studio (VS) in order to support specific features of hybrid dynamically and statically typed languages. Section 3 describes the implementation technologies used to customize the IDE, and its integration with the *StaDyn* compiler. In Section 4, we evaluate the programmer's performance improvement using our IDE, comparing it with similar approaches. Section 5 discusses related work, and the conclusions and future work are presented in Section 6.

2. A visual IDE for hybrid statically and dynamically typed languages

The proposed IDE can be applied to any object-oriented hybrid statically and dynamically typed language, such as Visual Basic, Objective-C, Boo, C# 4.0, Groovy 2.0, Fantom and Cobra. We have selected the *StaDyn* programming language, a hybrid static and dynamic typing language developed for research purposes [14]. *StaDyn* is an extension of C# 3.0 [16] that enhances the behavior of its implicitly typed local references (i.e., its `var` keyword). In *StaDyn*, the type of references can be explicitly declared, while it is also possible to use the `var` keyword to declare implicitly typed references. *StaDyn* includes this keyword as a whole new type (it can be used to declare uninitialized local variables, fields, method parameters and return types), whereas C# only provides its use in the declaration of initialized local references. An informal description of the language can be consulted in [14], while its static and dynamic semantics is detailed in [17].

In a previous prototype, we developed a first version of an IDE for the first implementation of the *StaDyn* language [18]. This first prototype was implemented as a plug-in for VS 2008. The one presented in this paper provides new features for both VS 2010 and 2012, using the Managed Extensibility Framework (MEF) included in the .Net Framework 4.0 (Section 3). The present IDE provides numerous new functionalities, representing a new contribution (Section 5 details the differences).