Dynamic Languages and Applications
Report on the Workshop Dyla'07 at ECOOP 2007

Alexandre Bergel[1], Wolfgang De Meuter[2], Stéphane Ducasse[3], Oscar Nierstrasz[4], and Roel Wuyts[5]

[1] Hasso-Plattner-Institut, Germany
Alexandre.Bergel@hpi.uni-potsdam.de
[2] Vrije Universiteit Brussel, Belgium
wdmeuter@vub.ac.be
[3] University of Savoie, France
stephane.ducasse@univ-savoie.fr
[4] University of Bern, Switzerland
oscar@iam.unibe.ch
[5] IMEC & Université Libre de Bruxelles, Belgium
roel.wuyts@ulb.ac.be

Abstract. Following last two years' workshop on dynamic languages at the ECOOP conference, the Dyla 2007 workshop was a successful and popular event. As its name implies, the workshop's focus was on dynamic languages and their applications. Topics and discussions at the workshop included macro expansion mechanisms, extension of the method lookup algorithm, language interpretation, reflexivity and languages for mobile ad hoc networks.

The main goal of this workshop was to bring together different dynamic language communities and favouring cross communities interaction. Dyla 2007 was organised as a full day meeting, partly devoted to presentation of submitted position papers and partly devoted to tool demonstration. All accepted papers can be downloaded from the workshop's web site.

In this report, we provide an overview of the presentations and a summary of discussions.

1 Workshop Description and Objective

The advent of Java and C# has been a major breakthrough in the adoption of some important object-oriented language characteristics. It turned academic features like interfaces, garbage-collection and meta-programming into technologies generally accepted by industry. But the massive adoption of these languages now also gives rise to a growing awareness of their limitations. On the one hand, researchers and practitioners feel themselves wrestling with the static type systems, the overly complex abstract grammars, the simplistic concurrency provisions, the very limited reflection capabilities and the absence of higher-order language constructs such as delegation, closures and continuations. On the other

© Springer-Verlag Berlin Heidelberg 2008

hand, dynamic languages like Ruby and Python are getting ever more popular. Therefore, it is time for academia to move on and to help pushing such languages into the mainstream. On the one hand, this requires us to look back and pick up what is out there in existing dynamic languages (such as Lisp, Scheme, Smalltalk, Self,...) to be recovered for the future. On the other hand, it requires us to further explore the power of future dynamic language constructs in the context of new challenging fields such as aspect-orientation, pervasive computing, mobile code, context-aware computing, etc.

The goal of this workshop is to act as a forum where we can discuss new advances in the design, implementation and application of object-oriented languages that radically diverge from the statically typed class-based reflectionless doctrine. The goal of the workshop is to discuss new as well as older "forgotten" languages and features in this context. Topics of interest include, but are certainly not limited to:

– agents, actors, active object, distribution, concurrency and mobility
– delegation, prototypes, mixins
– first-class closures, continuations, environments
– reflection and meta-programming
– (dynamic) aspects for dynamic languages
– higher-order objects & messages
– ... other exotic dynamic features which you would categorize as OO
– multi-paradigm & static/dynamic-marriages
– (concurrent/distributed/mobile/aspect) virtual machines
– optimisation of dynamic languages
– automated reasoning about dynamic languages
– "regular" syntactic schemes (cf. S-expressions, Smalltalk, Self)
– Smalltalk, Python, Ruby, Scheme, Lisp, Self, ABCL, Prolog, ...
– ... any topic relevant in applying and/or supporting dynamic languages.

In addition to the organisers, the program committee of the workshop included:

– Johan Brichau (Universit catholique de Louvain, Belgium)
– Pascal Costanza (Vrije Universiteit Brussel, Belgium)
– Erik Ernst (University of Aarhus, Denmark)
– Robert Hirschfeld (Hasso-Plattner-Institut, University of Potsdam, Germany)
– Matthew Flatt (University of Utah, USA)
– Dave Thomas (Bedarra Research Labs, Canada)
– Laurence Tratt (King's College London, UK)

2 Content

This section describes the organisation aspects of the workshop. The accepted papers and workshop slides can be found on the workshop's website[1].

[1] `dyla2007.unibe.ch`

Contrasting compile-time meta-programming in Metalua and Converge – Fabien Fleutot and Laurence Tratt

Powerful, safe macro systems allow programs to be programatically constructed by the user at compile-time. Such systems have traditionally been largely confined to LISP-like languages and their successors. In this paper we describe and compare two modern, dynamically typed languages Converge and Metalua, which both have macro-like systems. We show how, in different ways, they build upon traditional macro systems to explore new ways of constructing programs.

This presentation raised several questions regarding differences with other macro mechanism such as the one of Lisp-like languages. Also some issues regarding hygienic were successfully addressed by the presenter.

Relevant references related to this work are:

- The Converge programming language[2] [5]
- Metalua[3]

Collective Behavior – Adrian Kuhn

When modelling a system, often there are properties and operations related to a group of objects rather than to a single object only. For example, given a person object with an income property, the average income applies to a group of persons as a whole rather than to a single person. In this paper we propose to extend programming languages with the notion of collective behavior. Collective behavior associates custom behavior with collection instances, based on the type of its elements. However, collective behavior is modeled as part of the element's rather than the collection's class. We present a proof-of-concept implementation of collective behavior using Smalltalk, and validate the usefulness of collective behavior considering a real-life case study: 20% of the case-studys domain logic is subject to collective behavior.

The need for an accurate comparison with C++ templates was a good point raised by the audience. This will be addressed in future work, which also cover a formal description of the semantics.

How to not write Virtual Machines for Dynamic Languages – Carl Friedrich Bolz and Armin Rigo

Typical modern dynamic languages have a growing number of implementations. We explore the reasons for this situation, and the limitations it imposes on open source or academic communities that lack the resources

[2] convergepl.org/

[3] `metalua.luaforge.net`

> to fine-tune and maintain them all. It is sometimes proposed that implementing dynamic languages on top of a standardized general-purpose object-oriented virtual machine (like Java or .NET) would help reduce this burden. We propose a complementary alternative to writing custom virtual machine (VMs) by hand, validated by the PyPy project: flexibly generating VMs from a high-level "specification", inserting features and low-level details automatically – including good just-in-time compilers tuned to the dynamic language at hand. We believe this to be ultimately a better investment of efforts than the development of more and more advanced general-purpose object oriented VMs. In this paper we compare these two approaches in detail.

This presentation was preceded with a very convincing demonstration. A small interpret for a reverse polish notation calculator has been implemented. Very aggressive optimisations resulted in an highly optimised generated compiler for this calculator. Discussions were mainly about VM performance, especially when compared with Hotspot. Implementing Java on top of PyPy in order to assess VM performance was suggested. More information about PyPy is available online[4].

On the Interaction of Method Lookup and Scope with Inheritance and Nesting – Gilad Bracha

> Languages that support both inheritance and nesting of declarations define method lookup to first climb up the inheritance hierarchy and then recurse up the lexical hierarchy. We discuss weaknesses of this approach, present alternatives, and illustrate a preferred semantics as implemented in Newspeak, a new language in the Smalltalk family.

Pros and cons for having explicit *self* and *outer* sends in presence of virtual classes were presented. Several questions were raised from the large audience. Some of them covered the need of virtual classes in presence of closure. Gilad's answer was that each completes the other.

The Reflectivity: Sub-Method Reflection and more – Marcus Denker

> Reflection has proved to be a powerful feature to support the design of development environments and to extend languages. However, the granularity of structural reflection stops at the method level. This is a problem since without sub-method reflection developers have to duplicate efforts, for example to introduce transparently pluggable type-checkers or fine-grained profilers.
>
> This demo presents the Reflectivity, a Smalltalk system that improves support for reflection in two ways: it provides an efficient implementation of sub-method structural reflection and a simplified and generalized model of partial behavioral reflection. We present examples that use the new reflective features and discuss possible future work.

[4] `codespeak.net/pypy/dist/pypy/doc/news.html` and `pypy.org`

A number of questions were raised concerning the memory overhead. This appears to be largely due to the architecture of VMs, which are bytecode based. AST compression is part of the future work.

Some work related to this presentation[5] are *Sub-Method Reflection* [2], *Unanticipated Partial Behavioral Reflection* [4] and *Higher Abstractions for Dynamic Analysis* [3].

AmbientTalk/2: Object-oriented Event-driven Programming in Mobile Ad hoc Networks – Elisa Gonzalez

> The recent progress of wireless networks technologies and mobile hardware technologies has led to the emergence of a new generation of applications. These applications are deployed on mobile devices equipped with wireless infrastructure which collaborate spontaneously with other devices in the environment forming mobile ad hoc networks. Distributed programming in such setting is substantially complicated by the intermittent connectivity of the devices in the network and the lack of any centralized coordination facility. Any application designed for mobile ad hoc networks has to deal with these new hardware phenomena. Because the effects engendered by such phenomena often pervade the entire application, an appropriate computational model should be developed that eases distributed programming in a mobile network by taking these phenomena into account from the ground up. In the previous ECOOP edition, we presented and demonstrated AmbientTalk, a distributed object-oriented programming language specially designed for mobile ad hoc networks.
> This demonstration showcases AmbientTalk/2, the latest incarnation of the AmbientTalk programming language which supplants its predecessor while preserving its fundamental characteristics. The language is still a so-called ambient-oriented programming language which allow objects to abstract over transient network failures. This demo will highlight the new design choices in AmbientTalk/2 and the rationale behind them. The most important ones are the adoption of an event-driven concurrency model that provides AmbientTalk/2 with finer grained distribution abstractions making it highly suitable for composing service objects across a mobile network, and the integration of leasing techniques for distributed memory management.
>
> The demo is conceived as a hands-on experience in using the main features of the language where we show and discuss the following:
>
> – The development of an ambient application from ground up that illustrates the simplicity and expressive power of AmbientTalk/2.
> – While developing the application, participants become gradually acquainted with AmbientTalk/2's concurrency and distribution object models as well as the dedicated language constructs to deal with partial failures, service discovery and distributed memory management.

[5] `scg.unibe.ch/Research/Reflectivity/`

– We demonstrate how ambient applications actually behave in a real-life context by showing the execution of a small yet representative application on several portable devices such as laptops and smart phones.

AmbientTalk/2 is available at `prog.vub.ac.be/amop` with documentation and examples.

This very convincing demonstration used a personal digital assistant to communicate to a laptop using a wireless communication protocol. AmbientTalk [1] proves to be more expressive than traditional programming languages, especially about error recovery.

3 Conclusion

Most of the presentations and discussions of Dyla'07 present extensions of traditional dynamic languages. For example Metalua augments lua with an expressive macro mechanism, Converge is a Python dialect, Newspeak a Smalltalk dialect, and AmbientTalk a Self-like language. Comments and encouragement expressed by the audience asserted that dynamic languages constitute a viable research area. Efforts for experimentation and prototyping are greatly reduced in presence of a dynamic type system.

Dyla'07 lived up to its expectations, with high-quality presentations and demonstrations. Discussion were lively and stimulating.

Acknowledgments

We wish to thank Michael Cebulla and Jan Szumiec for their precious support. We also wish to thank all the participants.

References

1. Dedecker, J., Van Cutsem, T., Mostinckx, S., De Meuter, W., D'Hondt, T.: Ambient-oriented programming in ambienttalk. In: Thomas, D. (ed.) ECOOP 2006. LNCS, vol. 4067, Springer, Heidelberg (2006)
2. Denker, M., Ducasse, S., Lienhard, A., Marschall, P.: Sub-method reflection. Journal of Object Technology 6(9), 231–251 (2007)
3. Denker, M., Greevy, O., Lanza, M.: Higher abstractions for dynamic analysis. In: PCODA 2006. 2nd International Workshop on Program Comprehension through Dynamic Analysis, pp. 32–38 (2006)
4. Röthlisberger, D., Denker, M., Tanter, É.: Unanticipated partial behavioral reflection: Adapting applications at runtime. Journal of Computer Languages, Systems and Structures (to appear)
5. Tratt, L.: The Converge programming language. Technical Report TR-05-01, Department of Computer Science, King's College London (February 2005)