VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematics, Statistics and Computer Science
Te Kura Tatau

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Project Proposal

March 28, 2009

Neil Becker

Supervisor: Associate Professor Thomas Kühne

Submitted in Partial fulfillment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

This document describes a proposed research project to upgrade the current DeepJava language definition by implementing fixes, adding new language features and making improvements to the typechecker.

Project Proposal

1.1 Objective

The aim of the project is to upgrade the current DeepJava language definition. This includes implementing fixes to its handling of inheritance and polymorphism; adding new language features; and making improvements to the typechecker.

1.2 Working Title

"Upgrading the DeepJava Language Definition"

1.3 Background

Many problem domains span multiple model levels. Traditional OO languages, however, are limited to just two: those of an *Object* and its *Class*. Because of this mismatch developers must use workarounds which introduce accidental complexity [1] to the model. DeepJava (DJ) [2] is a conservative extension to the Java language which adds support for multi-level programming, thus enabling a direct mapping from model to implementation.

As soon as a model has more than two levels the question arises – "what are the elements in the middle levels?" In DJ, middle level elements, called clabjects [3], have both a type-facet for the level below and an instance-facet of the level above. Top and bottom level elements may also be thought of as clabjects, only without instance and type facets respectively.

In traditional two-level models a class constrains the attributes and behavior of its instances one model level below. DJ takes this concept further with a mechanism called Deep Instantiation, in which a clabject may declare the 'potency' of parts of its class facet. In this way it can constrain not just its immediate instances, but instances several model levels below.

Additionally the DJ runtime supports dynamic type creation without compromising static typing. And DJ offers an alternative approach to genericity, using meta-types instead of upper bounds.

The current DJ language implementation it built upon the Polyglot compiler [4], with runtime definition and manipulation of Java classes enabled by Javassist [5]. The implementation suffers from several limitations: inheritance is not supported beyond a single level; substitutability doesn't always work as expected; and type-checking needs improvement in some areas (for instance, 'final' declarations are not enforced and usage checking is dynamic only). There is also scope to add language improvements such as constructors with potency, and to make syntax changes to improve program readability. Also, much can be done to assist further development: the code base is sparsely documented and unit tests are minimal.

The final result of this project will be an upgraded version of the DJ language definition which will facilitate further research on this novel language design.

1.4 Resources

Required resources are minimal:

- A computer with Java, Polyglot and Javassist installed.
- Access to the DJ source repository on Elvis.

1.5 Method and Plans

I plan to:

- Read the DJ literature.
- Understand the current DJ language definition.
- Document my understanding in the code base (add JavaDoc).
- Identify shortcomings of the current language definition; both those described above and potentially others from my inspection of the code.
- Implement improvements to the language definition.

1.5.1 Milestone 1

By milestone 1 I will have completed a literature survey and made progress towards improving the DJ documentation.

1.5.2 Milestone 2

By milestone 2 I will have reached some of the goals set out above and will have a clear picture of what can be done in the remaining time.

Bibliography

[1] Colin Atkinson and Thomas Kühne, "Reducing Accidental Complexity in Domain Models," *Software and Systems Modeling*, vol. 7, no. 3, pp. 345-359, July 2008.

[2] Thomas Kühne and Daniel Schreiber, "Can programming be liberated from the two-level style: multi-level programming with deepjava," in *OOPSLA*, vol. 42, 2007, pp. 229 - 244.

[3] Colin Atkinson and Thomas Kühne, "A Tour of Language Customization Concepts," *Advances in Computers*, vol. 70, p. 105–161, June 2007.

[4] Nathaniel Nystrom, Michael R. Clarkson, and Andrew C. Myers, "Polyglot: An extensible compiler framework for Java," in *Proceedings of the 12th International Conference on Compiler Construction*, April 2003, p. 138–152.

[5] Chiba Shigeru. Javassist. [Online]. http://www.csg.is.titech.ac.jp/~chiba/javassist/