NextGen Developer's Guide

Eric Allen
eallen@cs.rice.edu
Rice University
6100 S. Main St.
Houston TX 77005

February 1, 2003

1 Introduction

This document describes the high-level architecture and code layout of the NextGen prototype compiler at Rice University. The NextGen compiler was developed as an extension to the GJ compiler under special license from Sun Microsystems. This same compiler was extended independently by Sun Microsystems to form the JSR-14 prototype compiler, scheduled for inclusion in J2SE 1.5. In the process of developing NextGen, we have refactored the original GJ compiler substantially, and no attempt has been made to maintain compatibility with the JSR-14 source code. Nevertheless, the reader may find that some of the architectural features of NextGen described here are helpful when deciphering the JSR-14 code base (modulo class, package, and variable name changes).

Throughout this document, it is assumed that the reader is familiar with the GJ, NextGen, and MixGen language designs, as well as the published descriptions of how these languages can be compiled to the JVM so as to maintain compatibility with existing compiled binaries. Readers not already familiar with this material are referred to [10, 9, 3].

2 The NextGen CVS Repository

The NextGen source code is maintained under the `javaplt` CVS repository at Rice University, and is available to all members of Rice JavaPLT. You can access

this repository from any machine with access the the Rice CS network. To do so, set your CVSROOT to `<user-name>@cs.rice.edu:/home/javaplt/.cvsroot`. If you are accessing the repository remotely, set your `CVS_RSH` variable to `ssh`. Then perform the following steps to set up your environment for NextGen development:

1. In your home directory, create a new subdirectory called `javaplt`.
2. In your `javaplt` directory, enter the following commands:

```
$ cvs checkout bin
$ cvs checkout packages
$ cvs checkout public_html
$ cvs checkout java/<your-plaform>
```

References

[1] O. Agesen, S.. Freund and J. Mitchell. Adding Type Parameterization to the Java Language. In OOPLSA'97.

[2] D. Ancona and E. Zucca. A Theory of Mixin Modules: Basic and Derived Operators. Mathematical Structures in Computer Science, 8(4):401–446, 1998.

[3] E. Allen, J. Bannet, R. Cartwright. First-class Genericity for Java. Submitted to ECOOP 2003.

[4] E. Allen, J. Bannet, R. Cartwright. Mixins in Generic Java are Sound. Technical Report, Computer Science Department, Rice University, December 2002.

[5] E. Allen, R. Cartwright, B. Stoler. Efficient Implementation of Run-time Generic Types for Java. IFIP WG2.1 Working Conference on Generic Programming, July 2002.

[6] E. Allen, R. Cartwright.The Case for Run-time Types in Generic Java. Principles and Practice of Programming in Java, June 2002.

[7] D. Ancona, G.Lagorio, E.Zucca. JAM-A Smooth Extension of Java with Mixins. ECOOP 00, LNCS, Spring Verlag, 2000.

[8] J. Bloch, N. Gafter. Personal communication.

[9] R. Cartwright, G. Steele. Compatible Genericity with Run-time Types for the Java Programming Language. In *OOPSLA '98*, October 1998.

[10] G. Bracha, M. Odersky, D. Stoutamire, P. Wadler. Making the Future Safe for the Past: Adding Genericity to the Java Programming Language. In *OOPSLA '98*, October 1998.

[11] M. Flatt, S. Krishnamurthi, M. Felleisen. A Programmer's Reduction Semantics for Classes and Mixins. Formal Syntax and Semantics of Java, volume 1523, June 1999.

[12] Martin Odersky and Philip Wadler. Pizza into Java: Translating Theory into Practice. In *POPL 1997*, January 1997, 146–159.

[13] Sun Microsystems, Inc. JSR 14: Add Generic Types To The Java Programming Language. Available at `http://www.jcp.org/jsr/detail/14.jsp`.