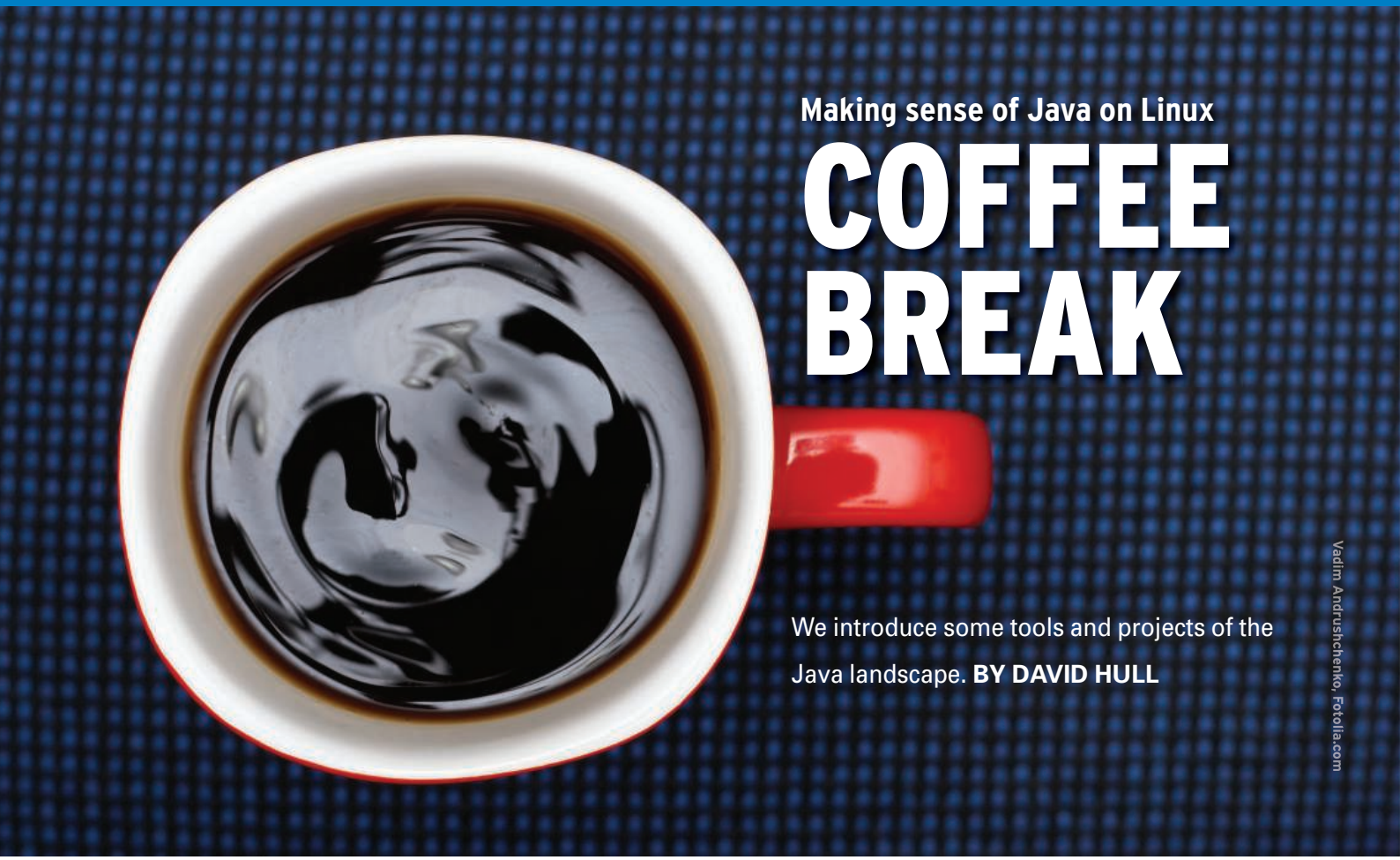

Making sense of Java on Linux

COFFEE BREAK

We introduce some tools and projects of the Java landscape. **BY DAVID HULL**

Vadim Andrushchenko, Fotolia.com

Linux is a free operating system. Java is one of the most popular platforms for free software. Running free Java software on Linux should be easy; yet, until fairly recently, anyone who wanted to run Java on Linux faced a dilemma. On one hand, you could use Sun's own Java environment, and you'd be guaranteed compatibility (at least with other Sun-based deployments), but you would be using non-free software. On the other hand, you could go with any of literally dozens of free offerings, ranging from someone's PhD thesis to major projects with release schedules and full-time staff, but these alternatives would lead to questions of compatibility under Sun's Technology Compatibility Kit (TCK) test suite.

This situation improved considerably in late 2006 and early 2007 when Sun began releasing its Java Development Kit (JDK) under a free license. Because Sun relied on other vendors for some parts of the JDK, it couldn't release them as free software. The IcedTea project [1] has filled in these missing pieces, both for JDK 5 and JDK 6, by bringing in appropriate parts of the GNU Classpath [2] (Figure 1). Because Sun has also released its HotSpot JVM [3] under an open source license, it is now possible to run Java on a completely free, TCK-compliant system.

Because the open source community still supports several other completely free choices that aren't officially blessed, the situation is still a bit complex. To run Java, you need some way of executing Java bytecodes, typically but not necessarily, with a Java Virtual Machine (JVM), and you need an implementation of whatever libraries you are using (together, this is more or less what Sun calls a Java Runtime Environment [JRE]). In this article, I will walk you through the major choices for JREs.

Which JVM?

The JVM is the component that actually runs your Java application. Because Java programs are represented as a sequence of bytecodes, the simple way to run a Java application is to interpret each bytecode and do what it says. This simple method is much too slow to be practical, so modern JVMs typically include a Just In Time (JIT) compiler that translates Java bytecodes to native machine code.

When Java was introduced, the idea of a new object-oriented language with a published specification and a major vendor behind it was catnip to programming language researchers, resulting in a burst of JVM development. Literally dozens of projects sprang up. Most of these projects had petered out by 2003 or so, but a few stuck around and gained wider acceptance.

Sun's HotSpot JVM comes in client and server versions. Both versions are free. The client version is tuned for shorter running applications, whereas the server version is tuned for long-running servers. The main difference is that the server version puts more effort into optimizations that have a greater effect with longer execution times. An application running on the client JVM will typically start faster but might run slower over time. HotSpot is the JVM shipped with OpenJDK [4] (Figure 2). At this writing, the latest point release is dated February 11, 2009.

JamVM [5], an alternative virtual machine developed by Robert Lougher and first released in 2003, is optimized for small size and fast startup time. The exe-

Work in Progress

The components discussed in this article are under active development. Statements about which versions are available, or packaged, or compatible with various other components reflect my best understanding as of this writing.

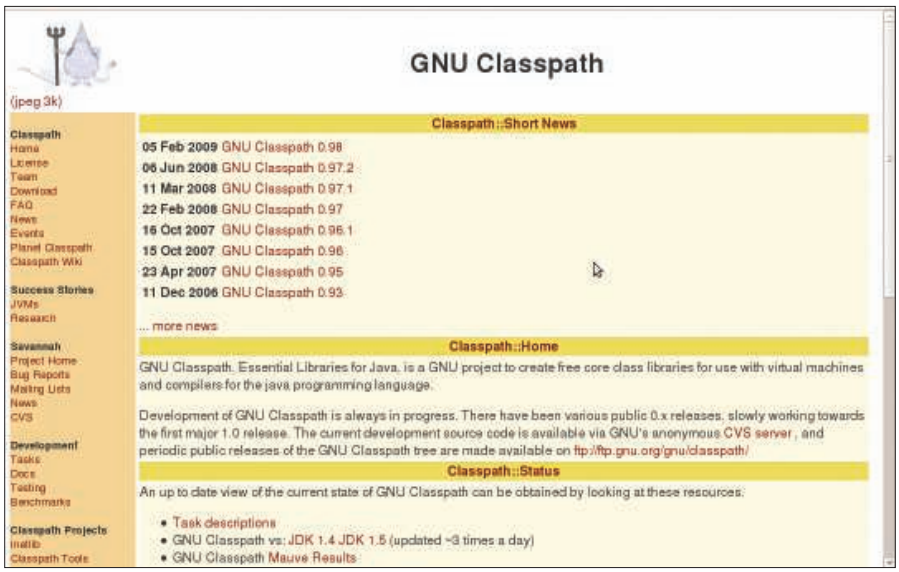

Figure 1: The GNU Classpath library is an important component of the open source Java environment.

cutable for the i486 and later Intel chips is around 180KB. JamVM is developed mainly on PowerPC, but it has been built and tested on ix86, ARM, AMD64, and MIPS chips. JamVM requires GNU Classpath for library support; according to the JamVM website, it is incompatible with OpenJDK. The latest release is 1.5.2, dated February 4, 2009.

Cacao [6] was originally developed in 1997 at the Vienna Institute of Technology as a faster alternative to Sun's JVM, which was a pure interpreter. Cacao instead compiles everything as it runs it. In this, it is more like the server version of HotSpot than the client. Cacao became an open source project in 2004. The latest release is 0.99.3, dated August 12, 2008.

Jikes RVM [7] (not to be confused with the RVM "recoverable memory" library) is the JVM produced as part of IBM's Jalapeño project. The Jikes VM is "meta-circular," meaning it is written in Java. The *R* in the name is for *research*, and RVM is aimed at demonstrating and trying out new and wonderful ideas in virtual machine technology. The project is very much alive – with a book just released about its architecture – but unlike the Jikes compiler, it does not appear to be actively packaged for distribution.

Kaffe [8] was originally developed in 1996 and was, for a time, the flagship product of Transvirtual Technologies. Transvirtual released Kaffe under the GPL in 1998, and it continues as an open source project. The Kaffe project is aimed at portability and small size. Kaffe has been ported to a large variety of platforms, ranging from small embedded devices to Linux, Mac, and Windows. The developers do not claim full Sun compatibility. In fact, the website goes so far as to explicitly disclaim Sun support. In particular, Kaffe does not support security features such as bytecode verification.

The SableVM project [9] aims to provide a highly portable and understandable JVM implementation. SableVM uses a sophisticated interpreter to achieve performance "near" that of a JIT. According to the website, the Sable project "has met its research goals and is not actively maintained anymore"; indeed, the last release was 1.13, dated December 2005. Nonetheless, SableVM is available in package form for major distributions.

Another option is to compile with the GNU Compiler for Java (GCJ) [10]. Did I say the JVM is what actually runs Java programs? Usually it is, but you don't have to take the usual route of compiling Java to bytecodes and then feeding them to a JVM that will most likely compile them even more. Instead, you can feed the bytecodes to the GCJ and produce native code directly. Also, you can use the GCJ to compile Java Archive (JAR) files to native code. The finished product will run like any other native executable, and a *libgcj* shared library even provides the standard library environment. This process is what used to be called "compilation," but in the context of a bytecode-based system like Java or .NET, it is called "Ahead Of Time" (AOT) compilation to distinguish it from "just in time."

Many of the earlier open source JVM projects are still around, and some of them might even work with modern Java systems. To use them, you'll have to build the code yourself and, quite possibly, get your hands dirty in the code.

In addition, you will find a number of more specialized JVMs aimed at embedded applications or other niches. They aren't included here, but if you're curious, the Kaffe site has an extensive list [11]. A virtual machine called IKVM [12] is even available for running Java on .NET, and it works with Linux. Tools like IKVM might be useful if you're developing for .NET, but for running ordinary Java applications on Linux, it's probably better just to run them in a standard environment.

Which Libraries?

Mercifully, the task of choosing libraries is much simpler than choosing JVMs because the open source user has fewer options. After all, writing a complete set of libraries is a much bigger task than writing a JVM. For example, the GNU Classpath project has 20 active developers, whereas a VM tool like JamVM has just one. And let's face it: Writing a new JVM with a cutting-edge optimizing JIT is a lot more fun than writing 18 different versions of *Arrays.fill()*.

Most Java libraries can be written in Java and will run on any compliant JVM, leaving only a small set of packages that must be customized for the JVM implementation. In practice, all of the JVMs I listed, except HotSpot, integrate with GNU Classpath. Several also integrate with OpenJDK. Even OpenJDK itself used GNU Classpath code to replace the proprietary sections that Sun couldn't open source.

Although Sun has opened its library code, it still tightly controls the process and compatibility tests (in particular, its TCK) that allow an implementation to claim full Java compliance. Because of this, it is generally not practical for open source projects to claim full Java compliance, with the special exception of OpenJDK. Naturally, an open source project, Mauve [13], was started that is aimed at addressing this by providing a free test suite for Java class libraries.

GNU Classpath was practically the

only game in town for a long time if you wanted to run Java in a completely free environment. Classpath began as the library for the Japhar virtual machine, but as the various JVM projects figured out that writing an independent standard library cuts into actual JVM development, the various efforts began to merge. As a result, GNU Classpath has been integrated with all of the free JVMs described in this article except for HotSpot, which uses OpenJDK.

Although GNU Classpath can't use Sun's TCK, it can be tested with Mauve. Unfortunately, because both Mauve and GNU Classpath are moving targets, I can't say whether Classpath is even Mauve compliant, although it seems safe to assume GNU Classpath is substantially Mauve compliant at any given moment. Mauve is becoming a better and better proxy for TCK over time, so there's a good chance that GNU Classpath will "just work" for your purposes. But if you really care, you'll have to check for yourself.

OpenJDK, or something substantially derived from OpenJDK, is the only choice if you want full Java TCK compliance in a completely free environment. OpenJDK is compatible with Sun's JDK 5, and OpenJDK 6 (not too surprisingly) is compatible with JDK 6. When coupled with HotSpot, OpenJDK provides the official, Sun-blessed open source Java environment.

Apache Harmony [14] is an Apache top-level project (as of 2006) aimed at providing an independent, free Java environment compatible with Apache license 2.0. The requirement of Apache compliance precludes GNU Classpath, which carries the GPL "linking exception." Exactly why and when this might matter is a bit subtle, but in some cases of interest to Apache, it does make a difference.

OpenJDK/OpenJDK 6 is free, but it depends on Sun, both because its main code base comes from Sun and because Sun's license requires any modified version to be "substantially derived" from it in order to be able to use the TCK to ensure compatibility. For example, if you wanted to use OpenJDK as a base, but replace half of the packages, you probably couldn't claim TCK compatibility.

Finally, Sun had not yet open sourced its environment when Harmony was initiated. Harmony is not a complete rewrite of all the Java libraries, in that many of the standard Java components (e.g., Apache's own XML libraries) are already Apache-compatible open source implementations. The Harmony team seeks to integrate existing packages when possible and write from scratch when necessary.

Harmony is currently still under development with no official stable release in sight, although the site assures you the team is "making steady progress." The latest stable build is 5.08M, dated November 13, 2008. To date, Harmony has been integrated with SableVM, Jikes RVM, Harmony's own VM interpreter, and other VM implementations. It is a bit unclear whether Harmony will end up TCK compliant. The Harmony FAQ states that it will, but Sun has not yet officially allowed access.

Figure 2: The OpenJDK project is based on Sun's own Java Development Kit.

What Packages?

Debian has stable packages for GNU Classpath, OpenJDK, and OpenJDK 6. OpenJDK 6 with HotSpot is the default Java package for Ubuntu. Kaffe, Cacao, and SableVM are also available. Kaffe requires GNU Classpath. The others can use either GNU or OpenJDK 6. Red Hat RPMs are available for GNU Classpath, OpenJDK/OpenJDK 6 with HotSpot, Kaffe, and Cacao. Fedora 9 ships with OpenJDK 6, and other distributions should be following suit if they haven't already.

What To Do?

The good news is that there is now an easy option for free Java on Linux: Use OpenJDK with HotSpot. This option is free as in speech and free as in beer, and it is certified compatible with Sun's usual offering. (For most purposes, it *is* Sun's usual offering. If you have a recent major distribution, chances are you're already running it.)

On the other hand, if you want to experiment with different Java environments, that shouldn't be too hard either. Stable packages are available for several of the JVMs described in this article.

If you want to experiment with a lesser-known JVM, particularly if it is one of the versions dating to the early 2000s or late 1990s and it was not described in this article, you probably need to roll up your sleeves. Most likely, you won't have OpenJDK available, and you might not have GNU Classpath. If the project doesn't support Classpath or OpenJDK, you might have to settle for a less than complete set of libraries. The good news is, you don't have to play around with these partial solutions unless you really want to – if you do, you probably won't mind wrangling a few Makefiles. ■

INFO

[1] IcedTea: *http://iced-tea.org/*
[2] GNU Classpath: *http://www.gnu.org/software/classpath/*
[3] HotSpot JVM: *http://openjdk.java.net/groups/hotspot/*
[4] OpenJDK: *http://openjdk.java.net/*
[5] JamVM: *http://jamvm.sourceforge.net/*
[6] Cacao: *http://www.cacaovm.org/*
[7] Jikes RVM: *http://jikesrvm.org/*
[8] Kaffe: *http://www.kaffe.org/*
[9] SableVM: *http://www.sablevm.org/*
[10] GNU Compiler for Java: *http://gcc.gnu.org/java/*
[11] JVM list: *http://www.kaffe.org/links*
[12] IKVM: *http://www.ikvm.net/*
[13] Mauve: *http://sources.redhat.com/mauve/*
[14] Apache Harmony: *http://harmony.apache.org/*