What's Brewing?

What's Brewing With Java

Java is the foundation of our next generation product, BBj™. Here, we track and analyze the developments taking place in the world of Java. In this issue, BASIS Vice President of Engineering, Kevin King, explains our decision to implement BBj on the Java Runtime Environment 1.3.

Why Wait for JRE 1.3? Hotspot!

BBj will require the version 1.3 of the Java Runtime Environment (JRE). A number of our Customers have asked why we've built BBj for JRE 1.3 rather than the currently available JRE 1.2.

There are several compelling reasons for requiring JRE 1.3.

- JRE 1.3 provides several enhancements to the GUI capabilities of the Java language that are needed to make some of our visual controls, specifically the list box and the grid, function correctly
- JRE 1.3 provides an enhanced mechanism for system calls that is needed for the SCALL verb to execute correctly
- JRE 1.3 provides significant speed improvements over JRE 1.2

Most of these 1.3 enhancements will not be noticeable to a BBj user. The one enhancement that will be significantly noticeable will be the speed improvement. The JRE 1.3 includes the Hotspot Just In Time (JIT) compiler, which is both a JIT compiler and a run-time code optimizer. It is through Hotspot technology that Java is approaching the execution speeds of C/C++.

Hotspot = Speed

To understand the effect of Hotspot, consider the following BBj program:

```
X = TIM
 FOR   I = 1 to 1000
          Z = ATN(3)
 NEXT I
 Y = TIM
 PRINT 3600 * (Y - X)
```

If you load this program and run it several times, you will observe something interesting. The second time the program is run, the execution time will be about half of the first execution time. The third time it is run, it will execute in about a third of the original time. After the third run, there may be some improvement, but the times will settle to about one fourth of the original execution time. Why does this simple program have such a variation in execution time? The explanation is that Hotspot is optimizing the underlying BBj code as it is running.

When a program is written in C/C++ and compiled with a C/C++ compiler, the result is a file containing machine code for a particular machine. When a program is written in Java and compiled with a Java compiler, the result is a file containing Java byte code. This byte code is a set of instructions that can be understood by the Java Virtual Machine (JVM) but not by the target machine itself. (Ed. Note: For a discussion of the JVM, see What's Brewing With Java, Quarter 2, 2000, Advantage.) The JVM must act as an interpreter that processes each byte code instruction by executing corresponding machine code.

Early versions of the JVM run Java programs slower than equivalent C/C++ programs. There are two reasons for the speed differential. The first reason is that each instruction expressed in byte code must be translated to machine instructions for the target machine and then the machine instructions executed. Often the translation takes as much time as the execution itself. The second reason is that modern C/C++ compilers analyze and optimize the source code during compilation. The technology of Java compilers is obviously not as mature as that of C/C++ compilers, and Java compilers do little, if any, optimization during the compile process. The Hotspot JIT compiler addresses these performance problems by doing JIT compilation and run-time optimization.

JIT Compilation

When Hotspot recognizes that a section of code is being executed repeatedly, it will cache the machine code that results from that entire code section. The next time Hotspot encounters the code section, Hotspot will use the cached machine code rather than re-translating the individual byte code instruction. Although Hotspot may not cache all the code being executed within a given application, it will cache the code that is most often executed.

Run-Time Optimization

When Hotspot recognizes that a particular set of cached machine code is being executed numerous times, it will analyze that code and attempt to generate an optimized code section to replace the original. In theory, the information that Hotspot gathers before doing this optimization allows Hotspot to do a more appropriate optimization than can be done by the static optimizer of the C/C++ compiler.

Our simple program is a very real example of the power of Hotspot. In testing BBj with JRE 1.3, the tremendous performance increases we observed made the decision clear. We want to give you that power in BBj right out of the box. We feel it was worth the wait to build BBj on JRE 1.3.

For more information on speed comparisons between Java and C/C++, JIT compilers, run-time optimization and Hotspot technology, visit these Web sites.

www.

http://www.aceshardware.com/Spades/read.php?article_id=153
http://www.research.ibm.com/journal/sj39-1.html
http://www.sun.com/research/kanban
http://java.sun.com/products/hotspot/whitepaper.html