CAN JAVA REPLACE C++ ON WINDOWS FOR ACCELERATOR CONTROLS? [*]

H. Nishimura, LBNL, U. C. Berkeley, Berkeley, CA 94720 USA

Abstract
We discuss the possibilities of using Java instead of C++ for accelerator modeling, simulation and controls, covering the items of run-time performance, availability of numerical libraries, migration from C/C++, link to C++ routines, and distributed objects. We will be presenting Java class libraries for modeling and simulation studies, on-line device access and operation at ALS on Windows NT 4.0.

I. Java and Accelerator Control Systems

1-1. Platform Homogeneity

Portability was not an issue of accelerator control systems in their design phase. They were always made homogeneous as far as the platforms were concerned. Diversity was introduced as a result of painful evolution over time. Once the systems became heterogeneous, portability then becomes an issue of accelerator control systems. It is being recognized as a key to keep the systems flexible enough to catch up with the rapid progress of computer technologies.

Generally speaking, Windows NT is superceding Unix workstations in various fields. Although accelerator controls systems are conservative, accelerator control systems will be adopting Windows NT for better performance at lower cost. Therefore we must figure out how to manage diversity in a Windows NT-dominant system.

When a platform evolves in its own way, it is not a trivial task to ensure portability. In the case of Windows, it has been adopting very novel technologies more rapidly than any other platform paying almost no attention to the portability.

On the other hand, portability is almost everything for Java. Our discussion here is to investigate the feasibility of adopting Java to recover the portability when we use Windows NT as our primary platform.

1-2. Applet and Application

Java has been in our sight only for a few years and has already established its identity with the emerging boom of the Internet. However, it has been almost exclusively used for creating Applets that have strong restrictions to access computer resources for security reasons. Therefore, the power of Java has not yet been fully utilized. It is only recently that we started using Java to create application programs. Java is becoming a candidate to replace C++.

1-3. Portability and Its Cost

The merit of using Java is portability on multiple platforms. On the other hand, the price can be slower execution speed. In addition, it requires learning of both the language and the libraries. Cost of training should not be underestimated. The issue is how to evaluate, balance and compromise these merit and demerit. We will discuss them in the next three chapters.

II. Portability

Portability is usually understood to be an ability of a program to run on multiple platforms. This definition might be fine for most of the cases, but we distinguish two kinds of portability: (1) portability at run time, and (2) portability at development-time. Then we discuss a third item: (3) portability over time.

2-1. Portability at Run Time.

Portability at run time is achieved by a Java byte code and the run-time environment that is a combination of a Java Virtual Machine (JVM) and a standard set of libraries. Java is practically the only solution available today to create portable programs with graphical user interface (GUI) and can be very suitable for control system programs on consoles.

On the other hand, it is important to realize that the Java language is not a part of the run-time portability. Any programming language can be accepted if it produces a byte code that can run on the run-time environment of Java, and we have already seen several examples [1].

It should be also noted that the portability of Java is not unconditionally desirable for industry. For example, unmanaged run-time portability may not be appropriate for copyright issue. In addition, it is easy to reconstruct a Java source code from a byte code by using a decompiler. That means that a Java source code is not secured. These problems will be obstacles for industry to adopt Java to replace C++.

[*] This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Material Sciences Division, U. S. Department of Energy, under Contract No. DE-AC03-76SF00098.

2-2. Portability at Development Time.

The development-time portability is for developers creating a run-time image for a given platform. The image can be platform-specific. We can take it a kind of cross-platform development capability that is not specific to Java. A good example is CodeWarrior Compiler [2] that supports multiple target platforms.

2-3. Portability over Time

If we are designing the system from scratch, portability is not a current issue. Instead, portability in the future becomes more important. That is, portability should be taken in the time axis as continuous availability over time. Therefore, it is important to carefully observe where the industries are moving and pick up a solution that is expected to survive for a reasonable period. In this context, Java seems to be the only solution for portability.

III. Performance

As Java uses an interpreter, and not a native compiler at run time, it is thought that Java executes much slower than native programs. The situation is improving due to the following two items: (1) new technologies for JVM, and (2) native Java compilers.

At the same time, it should be noted that the execution speed has not been an issue of the accelerator controls software at a level where Java would be adopted.

3.1 New technologies for JVM

Just-In-Time compiler (JIT) technology has improved the execution speed of JVM and that it rivals with the performance of native compilers. Our recent experience shows the cases where JIT supercedes the speed of a native code.

3.2 Native Compilers for Java

There are several kinds of native compilers of Java already available. The following is a list of Java development environments on Windows NT that include native Java compiles.

Symantec Visual Café for Java [3a]
Asymetrix SuperCede [3b]
Microsoft Visual J++ [3c]
IBM VisualAge for Java [3d]
Tower Technology TowerJ [3e]

The last two also support other platforms.

A native code is expected to run faster than JVM by sacrificing the cross-platform portability. It brings a better modularity of an executable image by linking libraries together. For commercial developers, it guarantees source code security by avoiding byte codes.

3.3 Numerical Calculation in Java

Although it is true that the execution speed is catching up with C++, it does not necessarily mean that Java is suitable for numerical calculations as pointed out by the paper [4]. Here we only emphasize that a lack of a standard numerical library is the most serious problem for us using Java for numerical calculations. There should be a basic set of standard mathematical libraries that cover linear algebra, integration and differentiation, statistics, equation solvers and optimization routines.

IV. Java and Other Languages

For practical software construction in Java, there is always a need for using routines written in other languages. (1) port at the source code level, or (2) external calls by Java Native Interface (JNI).

4-1. Port at the Source Code Level.

Assuming that the legacy code is written in C++, it can be a reasonable choice to port it at the source code level as Java is very similar to C++. In such a case, the following three items can be potential problems.

(a) Lack of operator overloading,
(b) Lack of default parameter values, and
(c) Lack of templates.

These problems are at the language specification level. Libraries cannot solve them. Possible solutions can be:

(a) Preprocess to a portable Java source code.
(b) Compile to a portable Java byte code.

Several preprocessors are being developed and posted on WEB[1]. One of the usable solutions that has been available is to use the extended Java compiler, Jump [5], which covers these features and produces a portable byte code.

4-2. External Calls by JNI.

Since JDK 1.1, Java Native Interface (JNI) provides a standardized way of linking Java to C++ routines. On Windows, JNI is a standard way of calling a dynamic link library (DLL) written in C++. JNI is portable in a sense that it does not assume any particular JVM on a given platform.

JNI plays a major role in creating a device control layer. Although it is possible to access C++ classes at higher level using JNI, it is preferable to limit the use of JNI at lower level where it becomes inevitable from a standpoint of code management.

V. Java for Accelerator Controls

It is true that there are problems with Java as we have discussed, the merit of adopting Java for accelerator control systems is rather obvious.

5-1. Portability on a Heterogeneous System

If a control system contains multiple platforms, Java gives enough portability to cover them. It is especially useful to gradually upgrade an existing system. The run-time performance of Java does not become an obstacle unless it is used at the real-time layer.

5-2. Standard Libraries

The Java distribution kit comes with standard set of libraries that cover wide variety of fields including graphics, database and networking. This is quite contrary to the case of C++ where we must select libraries very carefully. A potential problem is a lack of a library for scientific computing when a control system is model-based.

5-3. Device Access

It is possible to write a real-time layer entirely in Java by using Embedded Java when it is delivered. There are also some commercial products already available such as Jbed[6]. However, it is more realistic to assume that a real-time layer remains in C/C++ and publishes a JAVA API by using JNI. Although the users can do this, the manufacturers are expected to provide the JAVA binding.

5-4. Database

While a database is important for a control system, it is not seamlessly integrated into the system for most case. The situation with Java is similar but there is less diversity in available options to access the database. Java provides Java Database Connectivity (JDBC) that is equivalent to Open Database Connectivity (ODBC) for other languages. When the run-time performance of JDBC is not sufficient, there is a faster link available for major database systems, just as in the case of ODBC.

As the industries use databases much more than we do, there are already various commercial products available off the shelf. Many of them are component-based that use Java Beans.

5-5. Distributed Computing

In addition to the support for traditional networking, Java supports two of the modern distributed computing scheme, Common Object Request Broker (CORBA) and Remote Method Invocation (RMI), as a part of the distribution kit. They are also used in the industries mostly in conjunction with databases.

Enterprise Java Bean (EJB) is one of these examples that provide a framework for enterprise software systems. As EJB is too much for a database, it is not immediately useful for a control system but it can be a template for it.

Java development environments are to support such enterprise computing, which means an advanced support for CORBA and RMI. This immediately helps the development of control systems.

While CORBA is most accepted in the industries today, it is important to evaluate both CORBA and RMI. When a control system is highly Java-based, RMI will provide better over-all efficiency as it adheres to Java directly. It should be also noted that a RMI server allocates a thread for each client that gives a better response than CORBA when the total number of clients is less than 200 or so. A CORBA server recycles threads to handle thousands of clients that may make the response time unpredictable. We should be careful when we use the technologies developed for the business software.

One very interesting possibility is the use of JINI that is in a final stage of beta testing. As its very simple and flexible architecture, it can be very suitable for a control system.

5-6. Multi-tasking and Multi-threading

Multi-tasking is at the operating system level and involves OS-specific system calls that may not be well supported by Java. We must be extremely careful to confirm this issue. In case of Windows, only Microsoft Visual J++ compiler directly supports Win32 system calls.

Multi-threading is inside the Java programs. Although Java may use its own threading mechanism that is not native to the platform, it will not be a performance problem as far as we do not use it at a real-time layer.

5-7. Personal Resource

For any accelerator control system development, the bottleneck is always the availability of the controls software developers. The burden of staff training of Java may not be acceptable. On the other hand, if a control system is coherent with the industry trend, it becomes easier to introduce industry-based technologies, purchase commercial products and even to obtain human resources.

VI. EXAMPLES AT ALS

6-1. Local Device Access in Java

DMMobj is our C++ class library for the use on control consoles running Windows NT to access accelerator devices at the Advanced Light Source (ALS) [7]. It has been successfully ported to Java [8]. The C++ class is rewritten in Java and JNI is used at the lowest level. The port was quite straightforward.

6-2. Remote Device Access using RMI

DMMobj in Java was extended to be a distributed object by using Remote Method Invocation (RMI) and called RDMMobj[9]. It is strictly made read-only for safety reasons.

The ALS control system has a bandwidth of over 1000 read access per second to the online devices. Therefore, there is no need to perform a ganged access to a group of devices. An atomic access is always fine on the consoles. However, this scheme is not acceptable over the network because of the overhead of data transfer.

Therefore, we modified the RDMMobj so that it can perform block data transfer. It takes only 232 msec to read all the BPMs of the ALS storage ring over the network. In this case, it involves read and transfer of 192 R4 data. It takes only 1.21 msec to read a R4 data from an on-line device in case when 192 readings are grouped.

6-3. Modeling and Simulation in Java

We have also ported a subset of Goemon [10], a C++ class library for accelerator modeling and simulation, to get TracyJ in Java. This implements the standard 4x5-matrix formalism for linear optics calculations. The run-time performance of TracyJ is much better than we expected. Here is a comparison of run-time speed for two cases:

A. A step-by-step particle tracking of the ALS storage ring for 10,000 turns.

B. A dynamic aperture calculation of the ALS storage ring.

Both are basically a repetition of matrix and vector multiplication an were done on a PC with dual Pentium 2, 450 MHz, 128 MB of RAM running Windows NT 4.0 SP4.

	Case A	Case B
Java Interpreter	sec	sec
Sun Classic JVM 1.2	4.6	43.1
Win32 Native Java Compiler		
Visual Cafe 3.0	12.8	113.5
Tower J 2.22	8.2	67.7
Visual J++ 6.0	5.5	48.0
Win32 Native C++ Compiler		
Visual C++ 6.0	7.5	51.8
C++ Builder 3.0	8.8	46.0

Remember that this comparison was done as a part of our porting process of Goemon to Java. It is not intended to be a benchmark test to compare the run-time performance of Java and C++. Both versions of source codes are differently optimized by hand and there is no line by line correspondence. Each version must have room for better optimization. Nevertheless, we can safely claim that Java has become suitable for scientific calculations as far as execution speed concerns in case of accelerator modeling and simulation.

VII. CONCLUSION

As far as we have experienced by creating small examples at ALS, the run-time performance of Java on Windows NT 4.0 is satisfactory not only for accelerator control purposes but also for scientific computing. We believe that the use of Java should be always a part of options for any accelerator control system design when we pursue the portability to keep the system flexible enough to evolve over time. Java is especially suitable for continuous system upgrades if the system has become heterogeneous in spite of its original design.

ACKNOWLEDGEMENTS

The author thanks R. Hajima at U. of Tokyo for fruitful technicall discussion, A. Jackson and D. Robin at ALS for their encouragement.

REFERENCES

[1] R. Tolksdorf, “Programming Languages for the Java Virtual Machine”, http://grunge.cs.tuberlin.de/~tolk/vmlanguages.html
[2] Metrowerks Corp., Austin, TX
[3] a) Symantec Corp., Cupertino, CA. b) SuperCede, Inc., Bellevue, WA. c) Microsoft Corp., Seattle, WA.
d) IBM Corp., New York, NY. e) Tower Technology, Austin, TX
[4] W. Kahan and Joseph D. Darcy, “How Java's Floating-Point Hurts Everyone Everywhere”, ACM 1998 Workshop on Java for High-Performance Network Computing, March 1998. http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf
[5] D. Hoffner, “The JUMP Compiler”, http://ourworld.compuserve.com/homepages/DeHoeffner/jump.htm
[6] Oberon microsystems, Inc., Zürich,Switzerland
[7] A. Jackson, IEEE 93PAC, 93CH3279-7,1432, 1993.
S. Magyary, IEEE PAC93, 93CH3279-7,1811,1993.
[8] H. Nishimura, IEEE 95PAC, 95CB35843,2162, 1996.
H. Nishimura, LSAP-153, LBNL, 1993.
[9] H. Nishimura, LSAP-256, LBNL, 1998.
[10] H. Nishimura, LSAP-261, LBNL, 1999.