



Java Proving Itself Worthy for Defense Apps

As the military migrates away from Ada, Java is brewing to become the next dominant software technology for defense programs. Creating real-time Java specs is the final ingredient to enable soup to nuts acceptance of Java.

Jeff Child
Senior Editor

To the world at large, Java is perceived as just a nifty scheme for running applets on web browser pages. But to software programmers and system developers, Java is no less than a software programming revolution—a revolution that's eroding developer's interest in older languages like C, C++ and Ada. Java is a clean, well-defined language that incorporates the flexibility of C, the object-oriented structure of C++, and "write-once, run anywhere" portability.

Military equipment makers have had their eyes on that revolution, and they want in. Within the past few years, heavy hitters like Boeing, EADS (European Aeronautic Defense and Space Company), the U.S. Air Force and the U.S. Navy have expressed their desire to make use of Java for all levels of applications, from hard real-time safety-critical use up to non-real-time enterprise-like apps like command and control.

As a result, the various Java standards groups, like the J-Consortium and Java Community Process have put their rivalries aside in order to craft a spec that will win over even the most demanding hard real-time applications. In a nutshell, the goal is to craft a hard real-time Java core that's capable of sharing data with,

and running in conjunction with, all levels of Java.

Real-Time Java Core

With that in mind, The Open Group's Real-time and Embedded Systems Forum recently set out to establish standards for safety-critical and hard real-time mission-critical Java. The goal is to combine the strengths of the J-Consortium's "Real-

Time Core Specification" and the Java Community Process's "Real-Time Specification for Java." (See sidebar: "Groups Pool Efforts to Standardize Real-Time Java" for more details on the project's past and future.) The J-Consortium's Real-Time Core Specification outlines the general approach to be followed in this Navy-sponsored research and development effort. Table 1 shows what military appli-

Real-Time Core: What the Military Needs	Real-Time Core: What the Military Wants
Performance and footprint comparable to C and C++ with modern RTOS	Portable and scalable binary representations
Hard real-time determinism	Familiar Java syntax: leverages COTS Java development tools and Java expertise
Compile-time analyzers verify proper access to stack-allocated objects and enforce execution time limits on hard real-time components	Good object-oriented language
Simple micro-kernel architecture	Strong encapsulation, compile- and link-time type checking enables efficient large-scale development
Ada-like certifiable syntax and run-time	Secure dynamic loading (hot loadable device drivers for hot-swapped devices)
On-track to become ISO standard	Complements unadulterated traditional Java technologies to enable straightforward integration of traditional Java APIs

Table 1 With their heavy bent toward certification, determinism and standards, military programmers have some key needs when it comes to real-time software. But what they want centers on leveraging the richness of commercial Java technologies.

Groups Pool Efforts to Standardize Real-Time Java

The early work on the Real-Time Core was done by the J-Consortium, an open forum that develops Java technologies. It began as an alternative to the Sun-initiated Java Community Process by vendors and individuals that felt Sun's process wasn't sufficiently open and too centered on Sun Microsystems itself. The J-Consortium also crafted a subset of the Real-Time Core called the high-integrity profile, which provides safety-critical Java.

Within the last year and a half, the J-Consortium efforts melded with The Open Group. That process started when The Open Group was approached by key consumers of real-time Java technologies, including the U.S. Air Force and Boeing, asking their help to bring the real-time Java work done by the J-Consortium within the Sun umbrella. As a result, The Open Group became the arbitrator and facilitator to bring J-Consortium's work into the Java Community Process. An agreement was formed that's brought many of the original Sun real-time Java enthusiasts and J-Consortium technologists together. The Open Group will bring the Real-Time Core spec into the Java Community Process and the International Standards Organization (ISO) in parallel.

While the technology of the Real-Time Core spec is all but complete, there's expected to be some diplomatic and political wrinkles to be ironed out before the spec is finalized. The developer community and defense developers in particular want to avoid any standards war to erupt and cause fragmentation in the market.

The Real-Time Core spec will move a step closer to reality once NewMonics completes its prototype of the Real-Time Core. This month the U.S. Navy awarded a contract to NewMonics to implement that prototype. NewMonics' Kelvin Nilsen predicts that having the prototype in the hands of evaluators will go a long way toward accelerating the reaching of a consensus. "Right now it's more of a paper concept that people have great difficulty understanding," he says, "Within a year I think we will be in the final stages of setting the standard, and getting to a 1.0 version of the spec.

cations want and require from a real-time Java spec.

The Open Group is working with the U.S. Navy, NewMonics and many other commercial and defense interests to ensure that these standards will be endorsed both by Sun Microsystems' Java Community Process and by the International Standards Organization (ISO). In partnership with Aonix, an embedded development tool vendor and creator of the Ada language for the military and aerospace markets, and with funding support from the U.S. Navy and other prospective hard real-time Java users, NewMonics intends to provide the reference implementations for these emerging Open Group standards.

This integration of hard and soft real-time Java components will result in a new family of mission-critical products that will complement NewMonics' PERC virtual machine. With these developments, Java-based technologies will be able to meet the demands of performance- and footprint-critical applications common to defense, aerospace and telecommunications.

Navy's Open Architecture

Among the key motivations for the military's interest in Java is a drive to transition away from Ada. The feeling is that Java represents a modern and more commercially available technology than alternatives. The Navy, for example, is drafting their Navy Open Architecture Computing Environment (NOACE) to be the standard for all future software systems on Navy warships. That includes shipboard weapon systems, such as anti-aircraft cannon controls as well as avionics systems aboard naval aircraft. The standard calls for all new software to develop in either C++ or Java, and makes specific mention of moving away from Ada. They plan to continue to use Ada only as required to support legacy systems that have already been developed.

In his dealings with the drafters of the NOACE spec, Kelvin Nilsen, Chief Technology Officer at NewMonics, says he's hearing a very strong preference to use Java instead of C++. "The reason they don't exclusively recommend Java, is that it hasn't yet been proven in the breadth of

the domains relevant to the military and their requirements," he says. To do so it needs to support safety-critical hard real-time usage, which means response times in the nanosecond range. Nilsen is also Technical Chair of the J-Consortium, and the editor the Java Real-Time Core specification.

Until Java does prove itself as ready for hard real-time, Java is playing initially in the "soft" or "firm" real-time realm. There, response deadlines are measured generally in terms of tens of milliseconds. Those are middle tier applications that are PC-class hardware, 10s or 100s of Mbytes of memory and 200 MHz microprocessors. Generally that's low-end PC class, or rather PC-class of four years ago. Such software is often associated with Command and Control kinds of applications.

Moving Away from Ada

For its part, Boeing has also expressed a clear preference to move toward Java. Winner of the lead system integrator contract on for U.S. Army's Future Combat Systems (FCS) program, Boeing is farming out their FCS requirements and telling suppliers they want to use Java, they don't like C++ and they don't like Ada for any new system development. Many suppliers to FCS are therefore tasked to convert reams of Ada code over to Java.

There are some basic human resources reasons why Java is a preferred approach. Today's new graduates from college are 99% more likely to know Java than any other programming language. And among experienced programmers out in the workforce, more will tend to highlight Java on their resume rather than Ada. The ranks of true Ada gurus are probably comprised more of programmers near retirement than otherwise.

Java vs. C++

Another advantage Java offers is a broad selection of standard, portable and scalable libraries. That's where it has an edge over C++. While C++ has some good libraries, they're not portable—one set of libraries is needed with Windows, a different set is needed for Solaris, and yet another for Linux. Meanwhile, the complexity of C++ itself is a problem.

It's a big, complex language that many find difficult to master. Though experts can program effectively with C++, the problem is that when using C++ on a team project, everyone must be an expert, or problems arise. Unlike Java, C++ doesn't have enough barriers in place within the language to prevent inexperienced developers from polluting or infecting the whole system—one memory leak and everyone else's code breaks down.

Hierarchies of Real-Time Performance

Beyond the concerns about languages, military applications have sought Java as software technology that's capable of addressing different levels of real-time performance, with hard real-time capability at the heart of hierarchy (Figure 1). Such hard real-time Java is somewhat in its early days at this point. Developers are banking on the Real-Time Core specification to meet their needs.

NewMonics' Nilsen is careful to distinguish the difference between safety-critical and mission-critical real-time performance. In the mission-critical category are real-time Java virtual machines like NewMonics' PERC product. PERC enables response times in tens of milliseconds, and can even reliably support timing precision down to 100 μ s timing precision where necessary. Because it's not as fast as C or as small as C, it's a trade-off. But it allows the higher-level abstraction, easily integrated components, lower cost maintenance that Java provides.

The Real-Time Core spec meanwhile is a subset of Java designed to go a layer down from where PERC plays. It's designed to deliver a code size footprint and speed comparable to C or C++, depending on how you choose to develop code. If you use a lot of object-oriented abstractions in the Real-Time Core notation, the system runs at closer to C++ speeds. If the system is developed using a more static structure—then using a lot of object-oriented hierarchy—then code performs closer to C speeds. The Real-Time core is appropriate for 10 μ s timings and allows developing hard real-time code elements like software interrupt handlers and I/O device drivers.

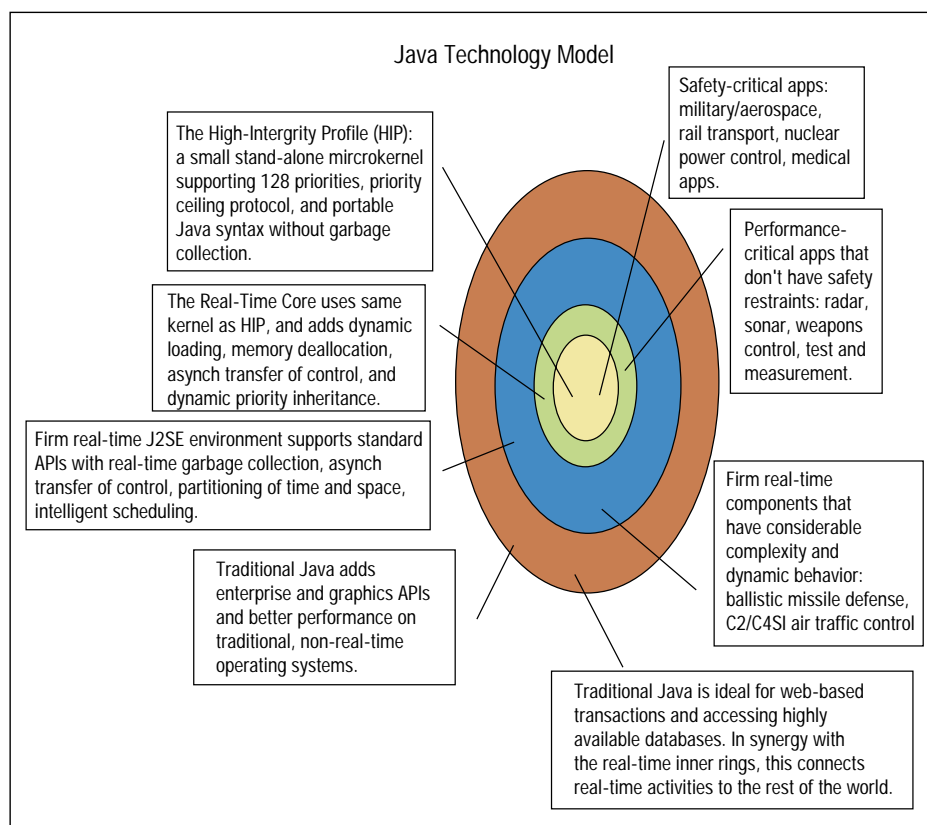


Figure 1 It's not enough to just have a high-performance hard real-time Java core. The real power is enabling the same Java syntax, tools and data to be shared among the different levels of real-time performance.

Compared to C, C++ and assembly language, and even compared to Ada, the Real-Time Core spec offers a much greater level of static analysis. In other words, it lets programmers perform compile time analysis or use development time analyzers to ensure that independently developed components stay within the realm of their responsibilities. It puts checks in place to make sure that independently developed code stays within its own memory and doesn't interfere with other memory or other CPU time.

Those checks can be done at development time because the Real-Time Core requires certain declarations to be inserted to make clear what the intent is of each component of code. The compiler verifies that they honor that intent. As a result, that component is easy to encapsulate, making it easier to assemble various components into a working whole. The Real-Time Core employs a simple micro-kernel architecture. That makes it easy to understand and fully portable. By following the standard, code will run the same whether

it's on PowerPC, MIPS, Pentium, Xscale or other processor.

The Real-Time Core is a subset of Java. There are certain classes—which are the standard Java libraries—that it doesn't support. There are also certain behaviors like automatic garbage collection that are in standard Java that aren't in the Real-Time core. Other behaviors, like the way initialization occurs, are different. Traditional Java initialization happens kind of on-the-fly while it's running. In a static hard real-time system, it makes more sense to do all initializations before it begins to run.

An Improvement Over C

A comparison with the C language reveals some of the benefits of the Java Real-Time Core notation. The notation is meant not as a replacement for C, but rather an improvement over it. For its part, C does not scale well to lots of lines of code and large teams of independent developers. It lacks the abstraction technology or the object-oriented compart-

The Softer Side

mentalization responsibilities. Meanwhile, C isn't as portable because data types can vary. In C, sometimes an integer is 16 bits, sometimes 32, sometimes 64 bits—and floating point handling can vary between implementations.

In contrast, the Java Real-Time Core always uses exactly the same data types. Its tasking definitions also stay the same whether you're running on VxWorks or Linux. Other schemes, like how to set a priority, what it means to wait on a queue, and so on are all nailed down in the specification. That ensures code will run the same no matter what platform it's on.

The Real-Time Core borrows Java syntax and can use a lot of the same Java tools. The Real-Time Core notation offers all the same object-oriented benefits of Java, such as encapsulation and the compile-time tools. That means programmers can work at a high level of abstraction with the assurance that components will fit together reliably when they're put together.

The fact that the Real-Time Core uses the same syntax as Java is compelling. That helps leverage the training and exposure programmers have with Java syntax. Developers will need to understand real-time concepts, but they can leverage what

they've learned from traditional Java and tools. Real-Time Core notation can use the same text editors, configuration management, and even the same byte code as standard Java. This makes it easy to marry Real-Time Core components to traditional Java software environments.

Developers can build layered hierarchies of software where the lowest layers are hard real-time—and in some cases even safety-critical code—and build traditional Java code above it. Data can be shared between the layers. In fact, many complex military and non-military systems have a requirement for such a hierarchy. Mission control systems have to talk to hard real-time components, for example, in order to determine a global position and send that data in an encrypted form across a wireless network. Meanwhile, an aircraft might need to expose information about its fuel supply and engine temperature and communicate that out to the air traffic controller.

Soup To Nuts Software

Once the real-time piece of the puzzle is added, Java covers that whole span of what's most important to military software developers. Using Java as a soup to nuts software solution that's portable

and scalable may ultimately help reduce the enormous software development costs typical in the military realm.

It reportedly costs the DoD \$400 to develop each line of code. This includes the costs of making sure the code properly specified before it written, meets the spec when written, and that whenever the code gets modified the software still works. Such careful steps are important for software that's expected to be in place over a long lifecycle. Today, when developers move code from one system to another system they typically have to rewrite over 80% of the code. With a soup to nuts Java strategy, that \$400 per line cost will go down because military developers won't have to rewrite so much code. ■■

Aonix North America
San Diego, CA.
(858) 457-2700.
[www.aonix.com].

Boeing Integrated Defense Systems
St. Louis, MO.
(314) 232-0232.
[www.boeing.com].

J-Consortium
[www.j-consortium.org].

Java Community Process
[www.jcp.org].

NewMonics
Tucson, AZ
(520) 323-9011.
[www.newmonics.com].

Sun Microsystems
Santa Clara, CA.
(650) 960-1300.
[www.sun.com].

The Open Group
San Francisco, CA.
(415) 374-8280.
[www.opengroup.org].