Java emerges as solution for military software modernization

By Dave Wood

The increasing complexity of military systems is creating pressure on software developers to modernize techniques focused on higher productivity levels and lower error rates. Java, which has provided high productivity and quality in desktop and mobile applications, offers a compelling alternative to C++ for next-generation mission-critical systems.

In recent months, major military and aerospace programs have begun turning to Java as a programming language of choice. Java is being used or actively evaluated in mission-critical applications for military aircraft, commercial airliners, battlefield wearable computers, naval combat systems, telemetry, unmanned aerial vehicles, and deep space exploration. The decision to migrate to Java for such complex and critical systems is not arbitrary, but results from the need to migrate mission-critical systems development in ways that emphasize highly productive COTS technologies.

Numerous large and complex military applications are under active development using Java. Some that can be publicly identified include Future Combat Systems, the Aegis combat system, and DDG-1000 (formerly DDX). Java is relatively new for real-time and embedded systems, but its current expansion into mission-critical development results from several years of ongoing efforts to develop COTS Java technologies to meet the specialized needs of mission-critical systems.

The growing importance of software modernization has been driven largely by the rapid evolution of hardware platforms. Hardware capacities and capabilities have been increasing exponentially for decades. Twenty or 30 years ago, the high cost and restrictive nature of hardware (processing, bus architectures, and memory) placed severe constraints on both the quantity and complexity of software that could be accommodated. Each year, more and more hardware barriers are eliminated, opening the floodgates for an inrush of software to take advantage of this blossoming hardware power.

The problem with C++

The expanding role of software presents two interrelated consequences in systems development: systemic fragility and prohibitive development/maintenance cost. The dominant programming languages used today in systems of all types are C++ and its predecessor, C. These languages still account for as much as 90 percent of mission-critical software developed today. Unfortunately, the design and characteristics of C/C++ present major hurdles in scaling systems to meet growing complexity challenges.

Increasingly disproportionate amounts of engineering resources are spent chasing bugs that could be prevented technologically. This forces developers to essentially reinvent the wheel when reuse is a more productive and reliable approach, and to fight to maintain fragile C++ software architectures against the tendency to degrade over time.

Chasing bugs is not a productive use of expensive engineering cycles. In the past, high hardware costs and restrictive capacities kept a lid on software costs. Today, software labor cost is becoming more of a limiting factor in what developers can achieve in their mission-critical systems and how quickly they can achieve it.

In this milieu, even a small increment in software productivity can have a very large impact in terms of decreasing costs, increasing functionality, and meeting critical deployment deadlines. Java provides an impressive track record as a productivity and quality enhancer.

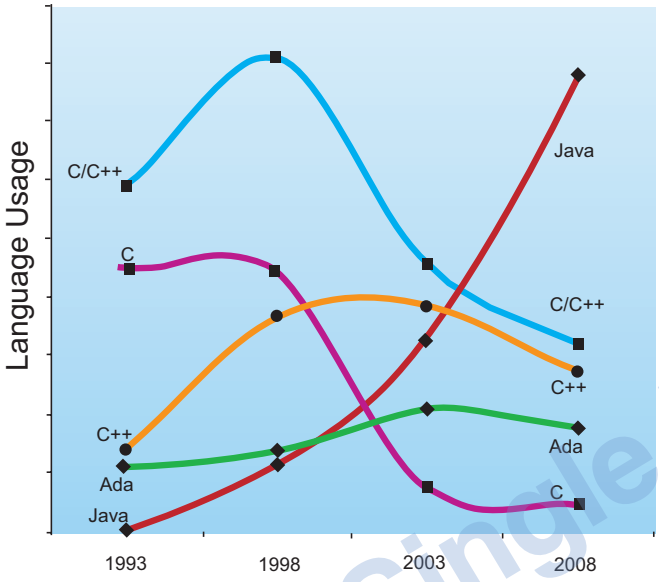
The continued use of aging methods and tools is a recipe for obsolescence and mission failure. For mission-critical systems, this problem is compounded by the fact that defense systems require the highest levels of reliability and longevity.

Java is not the only software technology better suited for mission-critical development than C++. Ada, which shares many of the technological benefits of Java, is used on many successful mission-critical programs and should continue to be used where viable. The advantages of Java over Ada are more contextual than technological: Java is vastly more popular commercially, resulting in a healthy vendor environment and extensive array of COTS tools and libraries.

Aegis program, photo courtesy of Lockheed Martin

New hires looking for Java

Virtually all computer science and software engineering students are emerging from universities with training and expertise in Java, compared to relatively few Ada-trained graduates. Because this pool of talent is driving future development endeavors, it makes sense to accelerate the process of modernizing mission-critical programs in that direction. Figure 1 illustrates the growth in Java use.



Usage Trends of Ada, C, C++, and Java

Data Source: *Programming Language Trends, An Empirical Study of Programming Language Trends*, Yaofei Chen, PhD dissertation, New Jersey Institute of Technology, August 2003

Figure 1

Rethinking modernization

Although many different notions of modernization exist, such as the mechanical transformation of legacy systems, our focus here is the modernization of emerging systems or subsystems and next-generation systems. To avoid repeating past mistakes, it is desirable to modernize not just a specific application, but also the entire thinking behind the software-development process. Doing this increases productivity, reduces error incidence, and heightens scalability and reuse in the development of new applications or new components of legacy applications. Whether modernizing individual components, a complete application, or the software development process, language selection plays a central role in modernization success.

Java is one compelling language alternative for use in modernizing mission-critical software. Although Java has not historically been considered appropriate for many mission-critical applications because of sluggish performance and lack of determinism, these roadblocks have been overcome in recent years. The emergence of Java technologies that enable high throughput computation and deterministic performance for embedded, real-time, and even safety-critical systems provides practical alternatives to C and C++ for new developments. Not only is Java easy to introduce to existing C++ contract houses (because of many similarities in syntax and terminology), Java also achieves superior scalability, portability, memory security, and simplicity compared to C++.

The problems inherent in C and C++ are well documented. According to recent market studies[1], more than half of C++ projects are running at least three to six months behind schedule. More than 40 percent of those delays are blamed on application complexity, a factor better addressed by Java than by C because of Java's superior modularity, scalability, and memory management techniques that allow programmers to avoid wasting time chasing insidious memory trashes and leaks. Twenty years ago, when most real-time systems were developed in assembly language on bare hardware, C offered an advanced language, representing a first-level abstraction away from the hardware. Eventually, C evolved into C++, adding object orientation and many other useful features, although many developers who use C++ compilers continue to confine themselves to the C subset of the language.

With rapidly growing software complexity, the importance of productivity improvement has become more acute. Java's features make it a good candidate to help prevent an exponential rise in software costs for the military.

A track record of productivity gains

As an alternative, published studies[2, 3] and real-world experience[4, 5] have shown Java programming is as much as two times more productive than C++, six times less burdensome in terms of error creation, detection, and correction, and much more portable across hardware architectures. Based on industry-wide productivity averages shown in Table 1 (derived from data at www.softwaretechnews.com/stn7-2/reifer.html), even modest productivity enhancements can shave hundreds of thousands of dollars off development costs and improve time to deployment by many months. Historically, mission-critical applications are produced at a rate of only 45-780 source lines of code per staff-month.

To those who have not considered the human cost of software development, the impact of productivity improvements on real-world cost savings is surprising. Based on observed Java productivity figures, lifetime savings for a moderate-sized

Application domain	Number of projects	Size range (KESLOC)	Avg. productivity (ESLOC/SM)	Range (ESLOC/SM)	Example application
Command and control	45	35 to 4,500	225	95 to 350	Command centers
Military – airborne	40	20 to 1,350	105	65 to 250	Embedded sensors
Military – ground	52	25 to 2,125	195	80 to 300	Combat center
Military – missile	15	22 to 125	85	52 to 175	GNC system
Military – space	18	15 to 465	90	45 to 175	Attitude control system
Trainers/Simulations	25	200 to 900	225	143 to 780	Virtual reality simulator
ESLOC = Equivalent Source Lines Of Code; SM = Staff-Month					

Table 1

mission-critical application can amount to several million dollars, while providing much higher reliability, superior scalability attributes, and faster deployment.

In the past decade, Java has become the language and platform of choice for desktop, Web, and mobile applications, widely displacing C and C++ in those application areas. The emergence of Java has substantially improved quality and productivity, thanks to the wide availability of Java-trained engineers and thousands of standardized and reusable off-the-shelf components. This development is making rapid deployment and reuse practical realities for the first time. Table 2 provides a small sampling of COTS libraries that can be used off the shelf with today's leading mission-critical Java development environments.

Tomcat	*Apache Web Server w/Servlets and JSP*
OpenJMS	*Java Message Service*
Oscar	*OSGi Framework and Bundles*
JSSE	*Java Secure Socket Extension*
JNDI	*Java Naming and Directory Interface*
Cryptix JCE	*Standard Cryptographic Algorithms*
mx4j	*Java Management Extensions*
OpenORB	*CORBA ORB*
Eclipse/SWT	*Eclipse IDE and SWT Graphics*
HSQLDB	*Lightweight SQL Database Engine*

Table 2

Bringing Java to mission critical

Beginning in 1997, the National Institute of Standards and Technology (NIST) coordinated the activities of an expert group in defining the requirements for the use of Java in real-time and embedded development. The NIST group – comprising more than 50 major technology companies including Sun, IBM, Microsoft, Aonix, Wind River, Motorola, and QNX – drew the conclusion that Java has the traits required to improve productivity and quality for real-time systems.

J-UCAS unmanned aerial vehicle, photo courtesy of Boeing

Following the conclusion of the NIST group's efforts, commercial companies undertook several years of specification, research, and development efforts to develop the first solutions for real-time Java developers. The primary concerns in creating an implementation were the real-time and embedded community's needs, including:

- Small memory footprint
- Fast response times
- High execution throughput
- Access to low-level devices
- Increasingly complex applications management
- Error incidence reduction
- Error detection and removal facilitation

These efforts resulted in the emergence of a variety of COTS technologies. For example, the introduction of predictable, real-time garbage collection within Java Standard Edition semantics has been successful in applying the modern advantages of Java to complex, mission-critical applications.

Ongoing efforts through The Open Group are driving emerging standards for deeply embedded, resource-constrained, and safety-critical application development through the Java Community

Java: Real-time use

Java's compelling traits

The NIST requirements group has identified the following Java traits (derived from NIST Special Publication 500-243) as the real-time requirements and motivation for Java's use by the real-time community:

- Java's higher-level abstraction increases programmer productivity
- Java is easier to master than C++
- Java is secure, keeping software components (including the Java Virtual Machine itself) protected from one another
- Java dynamically loads new classes
- Java is highly dynamic, supporting object and thread creation at runtime
- Java supports component integration and reuse
- The Java development process is conservative, using concepts and techniques that have been scrutinized by the community
- Java language and platforms offer application portability
- Java technologies support distributed applications
- Java provides well-defined execution semantics

Process based on specialized profiles of the real-time specification for Java. The first such technologies are now coming to market, providing a bridge to modernizing mission-critical applications requiring footprint and performance comparable to C and C++ applications, while retaining Java's scalability and reliability. New systems can now be developed in Java with integration to legacy components for expedience. Over time, legacy components, even those "close to the silicon," will be ported directly to Java code without capability or performance loss.

Forward-thinking program offices are now utilizing the new mission-critical Java technologies as a means to fight encroaching obsolescence and modernize both legacy and new applications. This is accomplished by tapping into the vast resources of COTS Java libraries and components, while taking advantage of the growing stream of Java-aware developers emerging from universities. Accelerating this trend is a crucial element in the battle to bring reliable systems to the field in an affordable and timely manner for years to come. CS

References

1. Venture Development Corporation, *Embedded Systems Market Statistics,* January 2006
2. Geoffrey Phipps, "Comparing observed bug and productivity rates for Java and C++," *ACM Software – Practice & Experience,* Volume 29, Issue 4 (April 1999): 345-358, ISSN: 0038-0644
3. Robert E. Wells, "Java Offers Increased Productivity," http://wellscs.com/robert/java/productivity.htm
4. Calix Case Study, http://www.javelocity.com/collateral/calix.pdf
5. Rob Hatcherson, Keith Holt, Stephen Tarter, "Using Java For Soft Real-Time Simulation," January 2004, www.zedasoft.com/UsingJavaForSoftRealTimeSimulation.html

Dave Wood, *VP of marketing at Aonix, has been involved with real-time and embedded software for more than 25 years. His activities have included analysis, design, coding, testing, research, and marketing technology in commercial avionics, telecom, defense, multimedia, software tools, and consumer electronics applications. Dave has worked for Lear Siegler, General Electric, SofTech, ComLogic, and the Software Engineering Institute at Carnegie Mellon University. He holds a BA from Kalamazoo College.*

To learn more, contact Dave at:

Aonix
5930 Cornerstone Court W, #250
San Diego, CA 92121
Tel: 858-824-0254 • Fax: 858-824-0212
dave.wood@aonix.com
www.aonix.com