

DRAFT

**Java and the
DII COE Real-time Kernel**

By

**DII COE Real-time
Technical Working Group (RT TWG)
and Integrated Product Team (RT IPT)**

23 June 1999

DRAFT

1. Executive Overview

The Defense Information Systems Agency (DISA) has started implementing the Defense Information Infrastructure (DII) Common Operating Environment (COE) kernel in Java for the 4.0 release. The RT community has concerns over the expected impacts of a potential Java implementation of a real-time COE Kernel on real-time systems development. These concerns are expressed in this paper.

This document summarizes our assessment of the viability of using Java as the implementation language for the DII COE RT Kernel.

For the reasons cited below, neither Java nor the anticipated real-time extensions to the Java platform ("real-time or RT Java") are considered to be capable of supporting the DII COE RT Kernel requirements in the target *runtime* environment *at this time*, and we recommend that real-time systems continue to use other high level languages to meet their operational requirements. Current Department of defense (DoD) efforts in the RT Java environment should focus on participation in the development of a robust international standard and conformance test suites, as well as the promotion of Commercial Off-the-Shelf (COTS) products and vendor support. The decision to use RT Java should be re-investigated after the specification(s) and initial conforming products exist and have had a chance to mature.

The remainder of this document presents the considerations affecting the proposed choice of Java as the implementation language for the RT DII COE kernel. Section 2 identifies the end-user (warfighter) runtime target system as the scope of our concerns. Section 3 summarizes general strengths and weaknesses of Java. Section 4 relates the status of the RT Java efforts. Section 5 addresses alternatives to Java as the initial implementation language for the RT kernel. Section 6 reiterates and elaborates the conclusions stated above. Section 7 provides our recommendations.

2. Scope of Concern Limited to Warfighter Needs

Our concerns regarding a Java implementation of the DII COE Kernel are focused on the warfighter, i.e., on the runtime environment of fielded real-time systems. We are not expressing any concern at this time with parts of the Kernel that will be used only in development or integration environments.

3. Current Capabilities of Java

The term "Java" is used in this paper to identify the current implementation of "Java" as we know it today. That is, in this context, Java refers specifically to Sun Microsystems Java. The specifications defining Java are openly available and Java is a de facto standard. However, Java is still not officially sanctioned by any recognized standards body and diverging implementations do exist. The term RT Java will be used as an abbreviation for

DRAFT

"the real-time extensions for the Java platform," when talking about the future capabilities that will be incorporated into Java that will allow using Java in more stringent real-time systems.

3.1. Java's Advantages

Java typically runs in a client-server environment. The Java Virtual Machine (JVM) allows the application to be downloaded from the server and executed on the client machine. The JVM, located on the client platform, interprets Java byte-code line by line each and every time a line of code is executed. Java applications can reside on the client machine; i.e., the client and "server" can be the same machine.

Java's main advantage relevant to the DII COE kernel is portability. Portability is achieved via the availability of a JVM on the desired set of platforms. The JVM is typically provided by the platform vendor or some other COTS vendor, and once it is available, Java byte code applications will run on the platform.

An enhancement to interpreted Java can be found in the implementation of "Just in time" (JIT) compiling. JIT compiling performs its function at program execution time, allowing the compiled code to remain cached in memory and reused based upon the commonality of each section of code. This caching technique enables *smaller* Java programs *with repetitive processes or sections* to execute quicker than they would in a pure interpretative environment.

Java's overall concept is that each client platform has its own machine-dependent Java interpreter (JVM) and/or JIT compiler, thereby eliminating the need to port the Java code to each platform. The only porting that needs to be done is the porting of the Java Virtual Machine and/or the JIT compiler. This implementation methodology tends to perform reasonably well in environments like those found on today's World Wide Web.

3.2. Java's Weaknesses for Real-time

3.2.1. Java's Unpredictable Memory Management Scheme

A key requirement of all real-time systems is that the performance of the system must be acceptably predictable; otherwise, there is no way to ensure that the system will meet its specified performance requirements. Java's method of implicit memory allocation and its dependence on garbage collection (GC) to reclaim memory result in very unpredictable timing behavior. While garbage collection algorithms have been designed that claim predictable real-time behavior, these algorithms are not typically used in Java implementations (which are typically not concerned with real-time behavior). The control of timing glitches due to GC as typically implemented is statistical, not absolute. Since GC is unpredictable, and prone to jamming when the system is hard-pressed, GC alone (without any other considerations) makes Java unsuitable for real-time applications. These

DRAFT

memory management problems are intrinsic to Java as well as any other language that uses garbage collection and would still be present even if the Java code was pre-compiled. These problems are not present when, as is already done with Java in time-critical applications, the memory management is performed statically in advance. In addition, both Real-time Java standardization activities (RTEG and RTJC) are considering mechanisms for achieving predictable garbage collection. However, in the absence of real-time Java today, the DISA plan is to implement the kernel using Java (*sans* real-time extensions).

3.2.2. Real-time Priorities and Priority Inversion

Process and thread priorities are a popular strategy for implementing real-time systems. Facilities for avoiding priority inversion (e.g., a low priority task blocking a high priority task) are also essential for the proper execution of real-time systems. Java does not support real-time priorities and does not address priority inversion.

3.2.3. Java's Threads Model

The threads semantics, as currently defined in Java, are not specified well enough to ensure that multiple vendors will implement them identically. Consequently, it will be necessary to validate the kernel applications for each and every platform. While this may be practical for the current set of non-real-time DII COE platforms, it is not a pragmatic approach for real-time systems in general.

3.2.4. "Footprint" Size

Most current JVM implementations are too large for many of today's real-time applications, especially embedded system applications, with JVM requirements for 4 to 10 Megabytes of CPU memory (RAM) being typical. This is not generally a problem with desktop platforms, but is a potentially serious problem for vehicle-based systems, which are often resource constrained, and is a severe problem for most DoD embedded systems, which typically have severe resource constraints. The impact of this Java characteristic would be to exclude many of the potential users of the RT COE.

3.2.5. RT Java Doesn't Exist

All manufacturers of current DII COE platforms (Sun, HP, and Microsoft) agree that Java is unsuitable for real-time applications. This is a primary motivation for their active participation and leadership in creating RT Java. At this point, however, the community has still not created and agreed on a technical specification for RT Java.. There are Java-like languages (c.f. NewMonics Perc) which support some aspects of real-time behavior. It should be noted that these languages (e.g. Perc) are single vendor products without the benefit of international standardization. However, since the specification effort has not been completed, no conforming RT Java implementations exist today.

DRAFT

3.2.6. Slowness of Java

Most implementations of Java are far too slow for real-time applications which often have millisecond, and sometimes microsecond, response time requirements [cf. Rockwell Collins' JEM chip as a counter-example]. Even for non-real-time uses, Java's lack of speed is a serious issue, as it is up to 10 times slower than pre-compiled Java. For example, as reported in the February 1999 edition of ACME Java Digest, an effort to re-implement Word Perfect in Java was abandoned because the Java version was too slow.

JIT compiled Java is also too slow for many real-time applications. JIT compiling is primarily advantageous for relatively small, repetitive tasks for which the compiled code remains cached, i.e., where the Java byte-code is compiled once and the compiled code is executed many times. For typical warfighter real-time applications, this will not be the case; typically the JIT compiler would be invoked each time the application runs, resulting in no significant improvement in execution performance. Note, however, that many mission-critical systems simply would not tolerate compilation on the fly. Java's performance might be acceptable if pre-compiled (not interpreted or JIT compiled). HP has a pre-compiled Java system, but this is not a typical implementation.

4. Current Status of RT Java

4.1.4.1. Community Process

Currently there are two independent and competing efforts underway to standardize the real-time functionality that will be incorporated into what will eventually become RT Java. One group is led by Sun and IBM and incorporates Sun's new "Community Process" for standards development. The other group is an industry consortium led by HP and Microsoft. There is no clear winner at this time, as both efforts are taking different routes towards obtaining a consensus based solution. It is not possible to know at this time whether these two "camps" will be reconciled. More importantly, it is impossible to predict what will be the detailed technical properties of whichever becomes the accepted standard for RT Java.

These competing requirements and implementation processes need to be monitored. Our best course of action at this time is to monitor, and perhaps participate, in both efforts, and work toward a single open and viable implementation of RT Java.

4.2. RT Java Development Schedule

An international standard for RT Java will not be approved in time to benefit COE release 5.0. It cannot be counted on even for COE release 6.0, simply due to the time required for specification development, approval of a proposed standard, and subsequent implementation and maturation of conformant RT Java COTS products. It is unlikely there will be a vendor RT Java specification before late 1999. At that point, the process

DRAFT

of obtaining the approval of an international standards body could begin in earnest. Work on a reference implementation might proceed in parallel with international standardization, with obvious risks. Consequently we are unlikely to see any vendor products until early 2000 to 2001 at the earliest.

5. An Alternative Solution

For real-time systems today, Java is not a viable alternative, and RT Java does not exist. ANSI "C" and Ada are proven, reliable languages for implementing real-time solutions for the warfighter. C++ may be acceptable for some applications. For real-time systems, the near-term solution seems obvious -- continue to use languages which are proven and reliable (Ada, C, C++).

Many of today's hardware device drivers have been written in C, and could easily be used by new mission applications, as well as new hardware devices. C is widely used for DoD real-time applications. Ada is also widely used within the DoD, and is the language of choice in some safety-critical commercial systems.

6. Conclusions

For the reasons outlined in section 3.2, Java is unsuitable for warfighter runtime environments, and therefore unsuitable for those portions of the DII COE RT Kernel which will be used in warfighter real-time systems (mission operations). RT Java is not a viable consideration at this time since it does not exist and will not be in the timeframe required to support DII COE version 5.0 development.

7. Recommendations

7.1. Recommendation for Near-term DII COE Kernel Implementation

Restrict the use of Java to those portions of the DII COE RT Kernel which will only be used in development and integration environments. Java should not be used to implement any portions of the Kernel which are needed for real-time operational missions. Instead, for Kernel functions which potentially will be used to support warfighter real-time needs, retain the current implementation language(s) in lieu of reimplementing these functions in Java.

7.2. Long Term Recommendations

Continue to monitor the progress of RT Java. The DoD should continue to track the progress and evolution of the RT Java programming language.
Periodically assess the status of RT Java. The RT TWG and/or RT IPT should perform an assessment of the status of RT Java. This assessment should be visible on the Web to TWG and IPT participants. It should be updated periodically (e.g., once or twice yearly).

DRAFT

Actively participate in the RT Java standardization effort. Subject to available funding, the DII COE RT TWG and/or RT IPT should actively participate in the two community efforts to produce RT Java specifications.