# Java for Real-Time Programming

Jan Lindström
Seminar on object architectures, Fall 1998
Email: jplindst@cs.helsinki.fi

Design of Java [2, 1] was originally motivated within Sun by the special needs of embedded system development and Sun is promoting the use of Java for embedded systems programming. Java, as it has been defined and implemented, lacks important capabilities necessary for the reliable development of portable real-time applications [9].

Computer programs that must execute within particular time constraints are said to be real-time programs or applications. Examples of real-time applications include voice synthesis, air traffic control, control of robots, telephone switching, and full-motion multimedia playback. Special implementation techniques are required of programmers who desire to enforce real-time constraints. In particular, programmers must determine the memory and CPU-time requirements of each real-time task independently, and then programmers analyze the collective workload of all the tasks that comprise the system workload.

One common technique for analyzing system workloads is known as rate monotonic analysis [3]. With rate monotonic scheduling, developers assign task priority in order of decreasing execution frequency. Tasks that execute most frequently are assigned the highest priority. Mathematical analysis of worst-case senario demonstrates that all of the tasks will complete their execution prior to their next period of execution as long as the combined workload represents less than 69% of the CPU's total capacity.

Developing real-time software is notoriously difficult and very costly. To exercise full control over real-time behavior requires extensive machine-dependent analysis. As a result, implementations of real-time systems are performed through a process of "black magic". Through human wizardy and experimentation, developers find the right combination of parameters that minimize cost while maximizing performance and compliance with real-time constraints. Many real-time programmers now do their development in C and C++. However, these languages do not provide mechanisms to allow programmers to describe real-time constraints. Consequently, programmers are required to enforce real-time requirements using combinations of compile-time analysis tools, pre-run-time measurements of application resource requirements, and run-time interaction with non-standard, real-time operating system services.

Note that development of real-time Java programs is especially difficult. The developer of a Java application has no idea how powerful CPU and how much memory will be available in the Java virtual machine environment in which the application is to run [9]. In real-time systems, application developers take responsibility for ensuring the software executes on schedule. In general, analyzing intertask timing dependencies is very difficult. And this analysis is especially difficult in the highly dynamic Java execution environment.

Java applications that will require more predictable real-time performance are already under development. These include real-time character animation, computer music synthesis, full-motion audio and video playback, and video conferencing. Java offers important high-level benefits to developers of traditional software. It would be desirable for real-time dialect of Java to improve upon the state of the practice in real-time development. In its current form, Java is not appropriate for the development of real-time software, because current Java definition and implementation include [9]:

- Unpredictable task execution times: Complicated control flow, absence of analysis tools, JIT translation, method caching, inlining, and other optimizations.
- Unpredictable memory requirements: Conservative garbage collection, failure to defragment, and complex library services.
- Unpredictable task interaction: Poorly defined priority system, priority inversion, non-incremental garbage collector, and complex dynamic workload.
- Weak vocabulary: Message, semaphore, interrupt, IO, signal, and timeout.

In this paper we present shortcomings of current Java [4, 5, 8, 9] implementations and a proposal for a real-time dialect of Java [6, 7, 10], which provides better support for real-time software development.

## References


[1] K. Arnold and J. Gosling. *The $Java^{TM}$ Programming Language. The $Java^{TM}$ Series.* Addison-Wesley Publishing Company Inc., 1996.

[2] Sun Microsystems Inc. The $Java^{TM}$ language environment: A white paper. Technical report, Sun Microsystems Inc., Mountain View, California, 1995.

[3] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[4] K. Nilsen. Issues in the design and implementation of real-time Java. *Java Developer's Journal*, 1(1):44–57, June 1995. http:// www. newmonics. com / WebRoot / technologies / java / RTJI.pdf.

[5] K. Nilsen. Embedded real-time development in the Java language. In *Proceedings of the Embedded Systems Conference*, pages 1–12, http:// www. newmonics. com / WebRoot / technologies / java / java-esce.pdf, April 1996.

[6] K. Nilsen. Real-time Java (v. 1.0). Tech. report, Iowa State University, Ames, Iowa, http:// www. newmonics. com / WebRoot / technologies / java / rtjava.api.pdf, January 1996.

[7] K. Nilsen. Real-time Java (v. 1.1). Tech. report, Iowa State University, Ames, Iowa, http:// www. newmonics. com / WebRoot / technologies / java / rtjava.api.v.1.1.pdf, February 1996.

[8] K. Nilsen. Invited node: Java for real-time. *Real-Time Systems Journal*, June 1997. http:// www. newmonics. com / WebRoot / technologies / java / java-rtsj.pdf.

[9] K. Nilsen. Adding real-time capabilities to Java. *Communication of ACM*, 41(6):49–56, June 1998.

[10] K. Nilsen and S. Lee. $PERC^{TM}$ real-time API (draft 1.3). Tech. report, NewMonics Inc., http:// www. newmonics. com / WebRoot / technologies / java / perc.api.pdf, July 1998.