IPP-HURRAY! Research Group

Polytechnic Institute of Porto
School of Engineering (ISEP-IPP)

To Ada or not To Ada: Adaing vs. Javaing in Real-Time Systems

Luís Miguel PINHO
Francisco VASQUES (FEUP)

HURRAY-TR-9904

January 1999

relatório técnico

technical report

THIS WORK IS PARTIALLY SUPPORTED BY FLAD UNDER PROJECT SISTER (Proj.471/97) AND BY DEMEGI/FEUP

To Ada or not To Ada: Adaing vs. Javaing in Real-Time Systems

Luís Miguel PINHO

IPP-HURRAY! Research Group
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340502, Fax: +351.22.8340529
E-mail: lpinho@dei.isep.ipp.pt
http://www.hurray.isep.ipp.pt

Francisco VASQUES

University of Porto (FEUP)
Rua Dr. Roberto Frias
4050-123 Porto
Portugal
Tel.: +351.22.5081702, Fax:
E-mail: vasques@fe.up.pt
http://www.fe.up.pt/~vasques

Abstract:
Ada is really an unfortunate Lady. After years fighting against C/C++ villains, her major lift-up (Ada 95) had brought up a promise of fortune. However, a new strong villain (Java) has appeared trying to end her struggle for survival.
Ada has now to fight with her own weapons. She will only prosper by her own merits. But two questions emerge. Do they exist? Are they better than Java's? Our opinion is that they do exist, and are not matched by any other programming language.

To Ada or not To Ada:
Ada*ing* vs. Java*ing* in Real-Time Systems*

Luís Miguel PINHO
Department of Computer Engineering
School of Engineering
Polytechnic Institute of Porto,
Rua de São Tomé, 4200 Porto, Portugal
e-mail: lpinho@dei.isep.ipp.pt

Francisco VASQUES
Department of Mechanical Engineering
School of Engineering
University of Porto,
Rua dos Bragas, 4099 Porto Codex, Portugal
e-mail: vasques@fe.up.pt

Abstract

Ada is really an unfortunate Lady. After years fighting against C/C++ villains, her major lift-up (Ada 95) had brought up a promise of fortune. However, a new strong villain (Java) has appeared trying to end her struggle for survival.

Ada has now to fight with her own weapons. She will only prosper by her own merits. But two questions emerge. Do they exist? Are they better than Java's? Our opinion is that they do exist, and are not matched by any other programming language.

Keywords: Real-Time Systems; Ada 95; Real-Time Java.

1. Introduction

Real-time systems impose specific requirements to programming languages [1]. Apart from the general software-engineering requirements, the interaction with physical devices, the concurrent nature of the system, and the predictability motivates the use of especially skilled languages.

The language must allow hardware interfacing. It must interact with conventional and special-purpose physical devices, which is a specific requirement of most of the real-time systems. Such interface must be handled in a flexible and reliable way.

Considering also that real-time systems are inherently concurrent, the programming language must provide support for multitasking activities. Hence appropriate scheduling and synchronisation mechanisms are a paramount requirement for the language. The concurrency must not only be supported by the Operating System services, since the compiler is not aware of the concurrent behaviour of the program.

As time predictability is a major issue in the design of hard real-time computing systems, the program timing analysis is a crucial step in the software development process. Therefore, a real-time program must allow an easy pre-run-time schedulability analysis [1].

Real-time systems have a long life expectation. Therefore, it is important that the software in it is easy to maintain, evolve, and port to several platforms. Also, software must be reliable and safe.

* This work was partially supported by FLAD (project SISTER 471/97) and by DEMEGI / FEUP

This is achieved by having unambiguous constructs and having strong type checking at compilation time. These characteristics bring an additional important advantage for the design of real-time computing systems: dependability. Although these are general “software-engineering” requirements, which must be considered in other types of applications, they are particularly relevant for real-time systems.

Ada 95 and Java are both contenders for overthrowing C/C++ from the real-time and embedded computing systems realm. However, the use of any programming languages must be driven by the requirements of the specific target application. It is a little bit distressing that the software programmers and systems engineers base the decision on the following: if *X* is the programming language that I know, why not use it?!

In the remaining of this paper, the term Ada is used to denote its 1995 standard [2]. When appropriate, Ada 83 will be used to refer its predecessor.

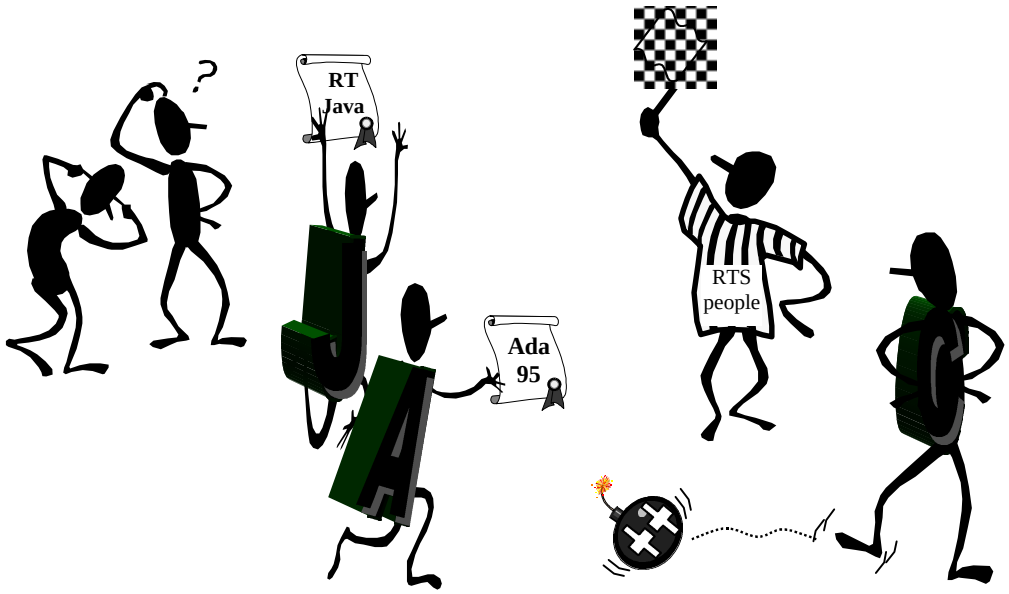

Figure 1 – The present

2. Java as a Language for Real-Time Systems

There is no doubt that Java has gained a lot of attention from the software programming community. Although initially aimed to the development of embedded systems, Java gained its reputation for its use in the field of heterogeneous, network-wide, distributed environments [3]. The Java platform provides a portable, familiar and simple object-oriented language. The syntax of Java derives from C++, with some modifications. The major are that, contrarily to C++, Java is fully object-oriented, has no pointers, and no explicit memory de-allocation. Therefore is simpler and safer than C++.

Java code is not compiled to the target machine, but translated to a standard portable byte-code representation, that can be executed on any Java Virtual Machine (JVM). This means that code is

completely independent of any specific target. This “write once, run anywhere” philosophy is the key element to its widespread use.

Java extensive number (and always increasing) of standard class hierarchies (Java API, Application Programming Interface) allow the development of platform independent software, ranging from multimedia to financial applications.

Concurrency, which is provided by the Java API (thus, not the language itself), makes possible the building of multithreaded applications. The multithread model is relatively simple, but there are still some undefined aspects: although providing thread priorities, their use depends on the underlying platform, hence preventing easy portability. More importantly, the *java.lang.Thread* class specification defines that: “Threads with higher priority are executed in preference to threads with lower priorities” [4], which is clearly insufficient for the development of real-time applications.

Synchronisation support of Java threads is based on the use of conditional variables (monitors). The program’s critical sections (code segments that access the same data from concurrent threads) must be synchronised using the *synchronised* keyword. The synchronised methods can use *notify()* and *wait()* to ensure that there is a lock contention for accessing the shared data.

The *notify()* method chooses one of the threads waiting on the monitor, and wakes it up. If there are multiple threads waiting, the Java runtime system makes no commitments about which will be the chosen thread. Another method (*notifyAll()*) can be used to wake up all the waiting threads. In this situation, the awakened threads compete for the monitor. Once again there is no knowledge of which thread will acquire the lock.

Java is heap-oriented, with a garbage collection mechanism, which frees the programmer from dealing with explicit memory de-allocation. The use of the Garbage Collector (GC), although preventing one of the greatest sources of errors in C/C++ code, imposes a non-deterministic timing behaviour of Java applications. This is another major drawback for its use as a programming language in a real-time computing system.

The Java language, the VM and the API specifications do not account for the real-time programming requirements [5]. However, there is an effort going on within the Java community for incorporating in the platform features suitable for its use in real-time systems (the so called Real-Time Java [6,7]).

The introduction of a pre-emptable GC, or the use of memory pre-allocation, can solve the problem of non-deterministic memory use. The extension of the API (Real-Time API) with new classes (like RealtimeThread) to support deterministic execution, and tightened thread scheduling semantics are also considered in order to make the language more appealing to the real-time community.

However, modifying deeply the Java language diminishes two of its strong arguments: portability and simplicity. The solution to this problem seems to be the specification of a core language, with one or more extensions (like annexes) intended to support the development of real-time systems applications.

3. Ada as a Language for Real-Time Systems

Ada 83 was designed to develop embedded systems applications. The DoD (Department of Defence) made an important decision at the time, recommending its use for the development of real-time systems. The main arguments were that the strong typing model of Ada 83 and its stack use leverage provided an effective detection of (most) errors at compilation time and an easy schedulability analysis. Additionally, by providing run-time constraints, errors, not detected at compilation time, can be treated by the program. Hence, Ada 83 soon became known for its reliability.

However, it also became evident that Ada 83 lacked some features particularly important for real-time embedded systems. Therefore, the language revision and its standardisation (Ada 95 [2]) brought a more open and extensible language without losing the intrinsic integrity of Ada 83.

A major evolution in the Ada language was the introduction of a new task model based on protected objects. This new task model allows a more efficient handling of shared data. Moreover, the data-oriented approach, introduced by the use of protected objects, is a step forward in the merge of concurrency and object orientation. Protected objects simplify the development of real-time systems, overcoming the disadvantages of the Ada 83 Rendezvous mechanism. With protected objects it is also possible to build asynchronous communication and mutual exclusion mechanisms.

An important advantage of Ada is that validation of compilers must be made by allowed institutions. A compiler targeting real-time systems must conform to the Real-Time Systems Annex and the Systems Programming Annex, in addition to conform to the core language.

The Real-Time Systems Annex provides the language with the required capabilities for performing pre-run-time schedulability analysis. This is achieved by the support of the Priority Ceiling Protocol, Priority Queuing and FIFO within each priority queue. Furthermore, it specifies a monotonic and accurate timing capability, and mechanisms for synchronous and asynchronous task control.

Ada mechanisms, such as for specifying the exact size and layout of objects, absolute variable addresses, together with an easy interface with other languages, simplify hardware interfacing.

Ada has undeniable characteristics, which provide strong mechanisms to guarantee timeliness requirements. These, added to the software-engineering concerns upon which Ada was developed, make it a truly real-time programming language.

4. Ada vs. Java

Internet and WWW are “hot topics”, consequently Java is *also* an “hot topic”. That seems to be enough to make Java a promising language. C and TCP/IP became *de facto* standards because people *de facto* used them. It will be no surprise if Java becomes a *de facto* language. It is therefore natural that Java will move into every type of applications (real-time systems included).

The main problem with Ada still is its complexity. Being a Pascal-like language with a strong type model, it is difficult to learn and not appealing to the majority of programmers. C/C++ and Java are undoubtedly easier to learn and use.

Another problem is the myth. Ada 83 had several drawbacks (compiler inefficiency, complexity) that prevented it from being widely accepted in the programming community. Java, on the opposite, is closely connected to the Web programming, and not too complex to learn, so it is extremely appealing to programmers.

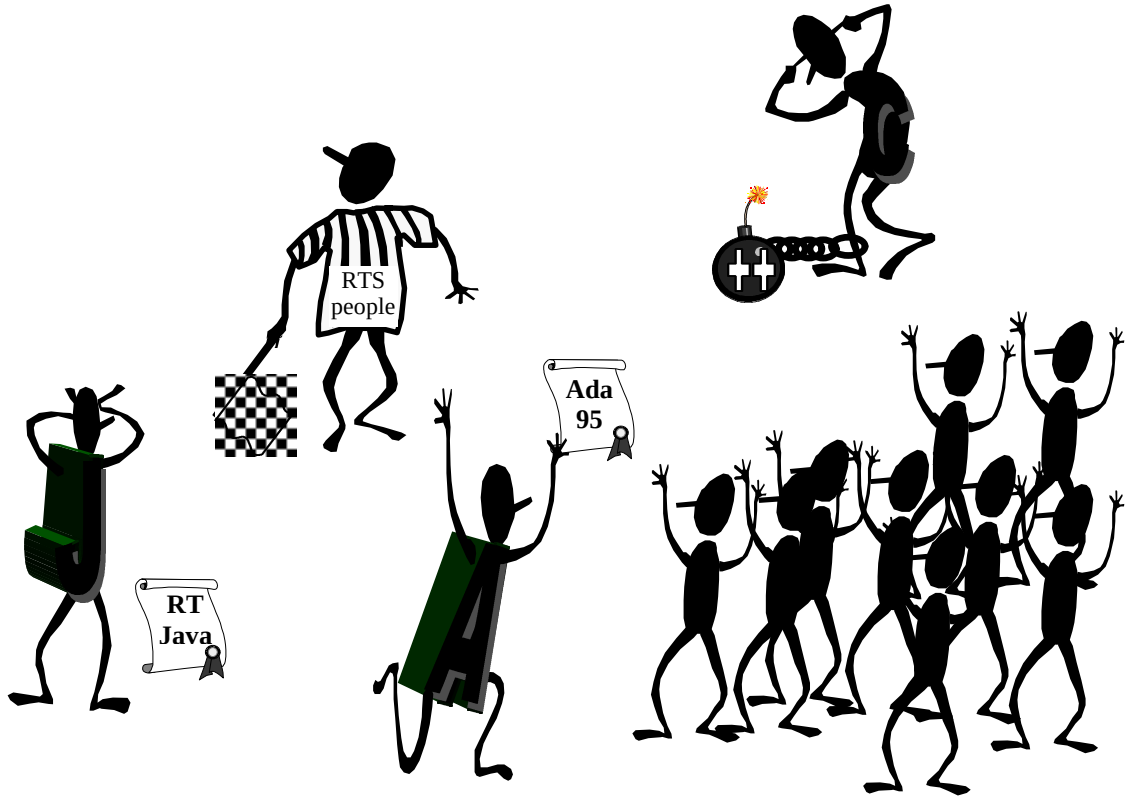

Figure 2 – The future

Ada offers a large number of features like object-oriented programming, concurrency support, hardware interfacing, strong type checking, generic units, etc., which can make the difference on the development of real-time systems. Its complexity is commonly referred as the reverse of the medal. However, it is our opinion that the easier software development process compensates the time needed to fully understand the mechanisms of the language.

Ada has a number of advantages over Java, such as the separation of logical interface from implementation, stronger compile-time type checking, true generic templates, enumeration types and higher-level synchronisation constructs.

The Real-Time Java process tends to incorporate[1] into Java features that are already present in Ada. If they are already present in Ada there is no reason for not using them today, in today's

[1] In fact, the process to extend Java into real-time systems (maybe we should call it Java 9X, or Java 200X) is somehow similar to the Ada revision process.

systems. Also, being Ada a safer language, it has some strong arguments for the Ada vs. Java fight, mainly in dependable real-time systems.

4.1 But Is Real-Time Java a Language for the Future Real-Time Systems?

Yes, but not *the* language. The software community has already seen some promises of an all-fit language that would conquer everything. But different systems have different set of requirements. An expert system has specific requirements that are better met by languages like Prolog, not Ada, C++ or Java. There is no such thing as an all-fit language. We have several programming languages with advantages and disadvantages, which must be weighted in each application domain.

Real-time systems include different broad areas, with different sets of requirements. In soft real-time systems with large dynamical evolution, such as multimedia or factory-level process control, Real-Time Java can, and probably will, take a major role. However, in systems with hard real-time requirements, where safety, reliability or availability are important concepts, Ada will continue to have a strong influence.

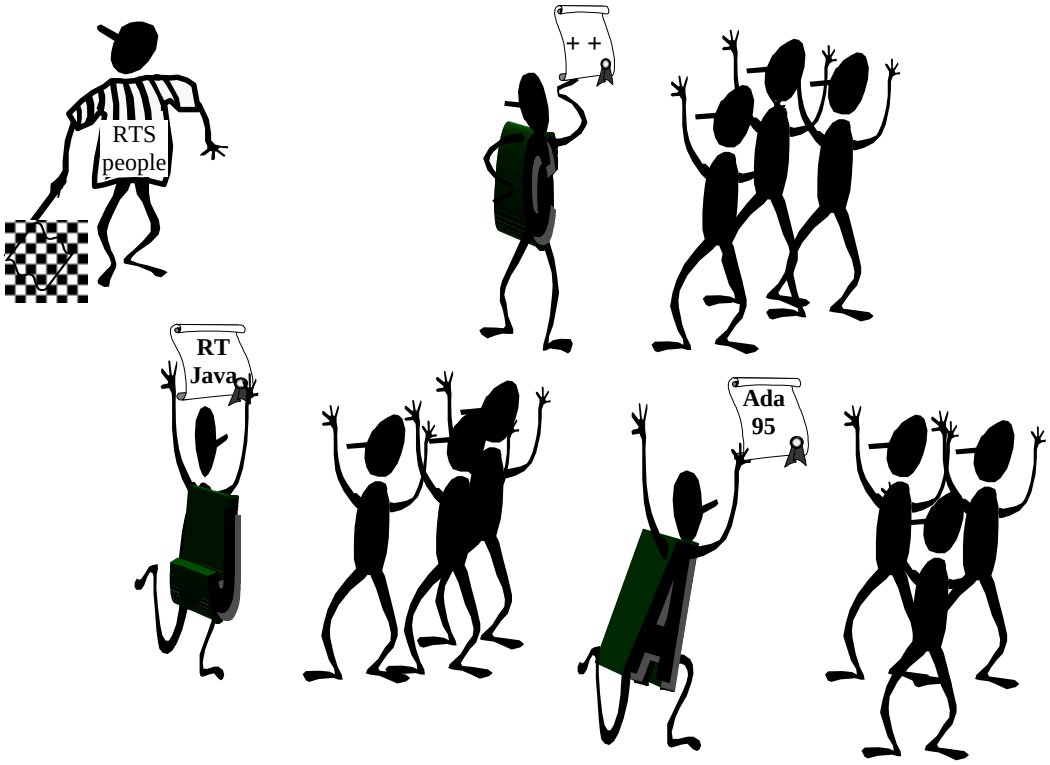

Figure 3 – The real future

5. Conclusions

This paper addressed Ada and Java (Real-Time Java) capabilities to support the development of real-time computing systems, presenting its advantages and disadvantages, and making some remarks on the future of these languages in real-time systems.

We feel that, although Java has been receiving a lot of attention, its specification does not present suitable mechanisms to support real-time systems. Real-Time Java tries to address this problem, incorporating in the Java language features that make it appealing to the real-time systems community.

However, Ada provides strong mechanisms to guarantee timeliness requirements, which added to its software-engineering oriented development, make it a truly real-time language.

As Java is a highly dynamic, heap-oriented language, based on C/C++ syntax, it is our opinion that Real-Time Java will always be less predictable and safe than Ada, which will remain as a more interesting language for hard real-time dependable applications.

However, the real-time systems and the Ada communities should welcome the appearance of Real-Time Java. It means that the C/C++ community finally admitted that this duet isn't good enough for real-time! But when will they admit that they need a truly real-time language?

6. Acknowledgements

The authors would like to thank Eduardo Tovar for his helpful comments on an earlier version of this paper.

7. References

[1] Stoyenko A. and Baker P., "Real-Time Schedulability -Analyzable Mechanisms in Ada9X", *in* Proceedings of the IEEE, Vol. 82, No 1, January 1994, pp 95-107

[2] ISO, Information technology - Programming languages - Ada, "Ada Reference Manual", ISO/IEC 8652, 1995

[3] Gosling J. and McGilton H., "The Java Language Environment: A White Paper", Sun Microsystems, May 1996

[4] Java Platform 1.1 Core API Specification, available at: h*ttp://www.javasoft.com/products/jdk/1.1/docs/api/packages.html*

[5] Uckun S. and Gasperoni F., "Making Java real-time", *in* IEEE Spectrum, December 1998, pp 22-23

[6] Nilsen K., "Real-Time Java (v. 1.1)", Iowa State University, Ames, Iowa, 1996, available at: *http://www.newmonics.com*

[7] The Requirements Working Group for Real-time Extensions for the Java Platform *http://www.nist.gov/rt-java*