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-9820

November 1998

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 approval. Ada has now to fight with its own weapons. She will only succeed by her own merit. But the question is, do they exist? Are they fitted to Real-Time Systems? We think that they do exist, and that they are more than suitable to Real-Time Systems programming.

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

Luís Miguel PINHO
Department of Computer Science
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 approval.
Ada has now to fight with its own weapons. She will only succeed by her own merit. But the question is, do they exist? Are they fitted to Real-Time Systems? We think that they do exist, and that they are more than suitable to Real-Time Systems programming.

Keywords: Ada 95; Real-Time Java.

1. Introduction

To meet the demands of Real-Time Systems programming, a language must cope with specific requirements, apart from the software-engineering general requirements placed to any kind of programming language [1].

As Real-Time Systems must interact with physical devices, a specific requirement for a Real-Time programming language is the ability to interface with conventional and special-purpose hardware devices. Such interface may be based on pooling or interrupting mechanisms, which must be handled in a flexible and secure way.

Considering also that Real-Time Systems are inherently concurrent, the programming language must provide support to multitasking activities, such as adequate scheduling algorithms and synchronisation mechanisms. The concurrency should not be supported only by the OS services, since the compiler isn't aware of the concurrent nature of the software.

As timing predictability is *the* major issue in a Real-Time Systems implementation, the program timing analysis is a crucial step in the software development process. Thus, a Real-Time programming language should be schedulability analysable.

Concerning software engineering generic requirements, as Real-Time Systems have a long life expectation, it is important that the software is able to maintain, evolve, and port across several platforms. Also, software must be reliable, so the programming language should be very secure, and with strong compile-time type checking.

Ada 95 and Java are both contenders to the realm of C/C++ applications: Real-Time Systems and Embedded Systems. Using any of these programming languages to support such kind of systems should be carefully evaluated before any decision. However, it seems that the only requirement that really matters is the following one: *XPTO* is the programming language that I know, so why not using it?!

In the following text, we'll be using the term Ada to refer to its 1995 standard. When necessary, Ada 83 will be used to refer the predecessor.

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 target to the embedded systems development, it was designed to meet the challenges of applications development in the context of heterogeneous, network-wide distributed environments [2]. The Java Programming language platform provides a portable, familiar and simple, object-oriented language. Its syntax is derived from C++, with some (major) modifications. It is a full object-oriented programming language, where some of the C++ features were removed to make a simpler and safer language.

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

Portability is one of the key features of the Java platform. 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. Its “write once, run anywhere” philosophy is the key element to its widespread use.

Java extensive (and always increasing) standard classes hierarchies (Java API) allow the programming of platform independent applications, from WWW animation to financial applications.

Concurrency support, which is provided by the API, makes possible the building of multithreaded applications. This multithread model is relatively simple, but there are still some undefined aspects: although providing thread priorities, its use depends on the underlying platform, defeating the portability issue. Besides that, the java.lang.Thread class specification defines that: “Threads with higher priority are executed in preference to threads with lower priorities” [3], which is clearly insufficient for real-time applications development.

Java threads synchronisation support is based in the use of conditional variables (monitors). The program 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 each value placed in the CubbyHole by the Producer is retrieved once and only once by the Consumer.

The notify() method chooses one thread waiting on the monitor and wakes it up. If there are multiple waiting threads, 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.

Although its embedded systems origins, the JAVA virtual machine and the API specifications still do not support real-time programming requirements [4]. The Sun approach is to create language variants (e.g. Embedded Java) to meet real-time requirements. However, this means that portability, which is one of Java’s most important features, is no longer supported.

Java is a heap-oriented language, with a garbage collection mechanism, freeing the programmer from explicit memory deallocation. The use of a Garbage Collector (GC), although preventing one of the greatest sources of errors in C/C++ code, results in the non-deterministic timing behaviour of Java applications, which is a major drawback for the Real-Time Systems development.

There is a trend in the Java community to incorporate features supporting Real-Time Systems programming (Real-Time Java [5]). The introduction of deterministic GC, the language extension to cope with hardware interfacing and the introduction of more elaborated concurrency mechanisms [4] are being considered in order to make the language more appealing to the real-time community.

However, modifying deeply the Java language would prevent its strong arguments: portability and simplicity. The solution to this problem seems to be the specification of a core language, with one or more annexes intended to support the development of Real-Time Systems applications.

3. Ada as a Language for Real-Time Systems

Ada was designed to develop embedded systems applications, in order to replace multiple languages used at the DoD. Ada 83 soon became know for its reliability. The strong typing model of the language allows the compile time detection of most errors, and the detection of the remaining by runtime constraints.

Its syntax being derived from Pascal, the Ada vs. Java discussion can be seen as a new battle of the C vs. Pascal war.

However, it also became apparent that the Ada 83 standard lacked some features necessary to its main target: Real-Time Systems development. The language revision and its standardisation (Ada 95 [6]) brought a more open and extensible language without losing the inherent integrity of Ada 83. It kept the Software Engineering issues, allowing for more flexibility in the software development process.

A major evolution in the Ada language was the introduction of a new task model based on protected objects. This new task model allows for a more efficient solution for the access to shared data. Moreover, the clearly data-oriented view brought by the protected objects fits in naturally with the general spirit of the Object Oriented paradigm. Protected objects simplify Real-Time applications development, outcoming the disadvantages of the Ada 83 Rendezvous mechanism. With protected objects it is also possible to build asynchronous communication and mutual exclusion mechanisms.

Asynchronous Transfer of Control is also a powerful language construct, allowing a task to receive events, without the need of polling or waiting cycles.

An important advantage of Ada is the required compiler validation. The targeting of any system being an Ada issue, induced the specification of a core language with some extra Annexes targeting special applications. For instance, a compiler targeting Real-Time Systems must conform also to the Real-Time Systems Annex and the Systems Programming Annex. The later covers access to machine code, interrupt handling and packages for general task identification and attributes. The existence of Ada mechanisms to specify the exact size and layout of objects for user data types and absolute variable addresses and its easy interface with other languages simplifies the job of hardware interfacing.

The Real-Time Systems Annex provides the language with the necessary capabilities for schedulability analysis, namely imposing the support of Priority Ceiling Protocol, Priority Queuing and FIFO within priorities.

Furthermore, it specifies a monotonic and accurate timing capability, mechanisms for synchronous and asynchronous task control and also tasking restrictions that, for instance, can impose a maximum number of tasks, no asynchronous control or no dynamic priorities.

The strong software engineering approach of Ada, together with its real-time systems capabilities, makes it a suitable language to real-time systems programming.

4. Ada vs. Java

Although Java first target was embedded systems, its main use is in Web programming, where it has great advantages. As Internet and WWW are "hot topics", Java is *also* an "hot topic". That is enough to make Java a promising language. We can point the case of C and TCP/IP, which became *de facto* standards because people *de facto* used them. It will be no surprise if Java assumes itself as the *de facto* language.

Having a great number of programmers, it is also natural that there will be a shift to Java in every type of applications, with Real-Time Systems among them. That is not sufficient to make a language suitable to any kind of application, but is sufficient to its utilisation.

The main problem with Ada remains in 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 language first-impression. 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, so it is extremely appealing to programmers.

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

Ada has a number of software-engineering related advantages to 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. In fact, the Java extension to support Real-Time Systems is somehow similar to the Ada revision process.

Real-Time Java (maybe we should call it Java 9X, or Java 200X) tends to incorporate 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 systems.

Also, being Ada is a safer language, it has some strong arguments for the Ada vs. Java fight, mainly in systems where fault-tolerance and safety are important.

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

Yes, but not *the* language of the future. 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. If we are doing an expert system, we'll probably use Prolog, not Ada, C++ or Java. There is not such thing such an all-fit language. We have several programming languages with their advantages, and disadvantages, that must be weighted in each application domain.

Real-Time Systems is a broad area, 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.

5. Using Ada for Real-Time Fault-Tolerant Programming

In this section, we describe some examples of Ada programming in Real-Time (and Fault-Tolerant) Systems, using some code developed for the Multi-μ [7] architecture.

The Multi-μ architecture is being implemented to study and develop software based fault tolerance mechanisms for Real-Time Systems, using the Ada language. It is based on the active replication of processing nodes, using Commercial Off-The-Shelf (COTS) components.

Each node (figure 1) has a real-time kernel, responsible for the multitasking environment and for the communication with other nodes. The application is built on top of the compiler library, to ensure abstraction from kernel implementation, and also on top of the Fault Tolerance Manager (FTManager), providing the fault tolerance abstraction.

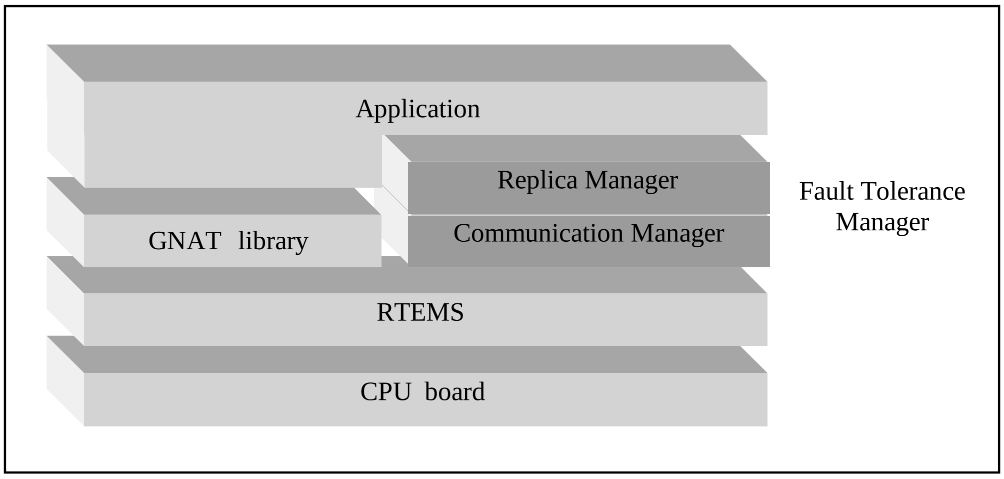

Fig. 1 – Multi-µ Node Architecture

The selected kernel is the Real-Time Executive for Multiprocessor Systems (RTEMS) [8] and the selected Ada compiler is GNAT [9]. RTEMS is a real-time kernel suitable for real-time applications as it implements the needed features (multitasking, multiprocessing, preemptive scheduling, intertask communication, priority inheritance, etc.). It has a modular architecture, and so it is possible for non-used features not to be integrated in the application code. The RTEMS tasking system will be used to support the GNAT run-time system, which is a work currently being done by RTEMS and GNAT people.

The RTEMS kernel provides communication links between nodes making use of queues. This mechanism can be used without knowledge of the physical distribution of the sender and receiver tasks, being a good framework for building replicated systems.

The FTManager is responsible for the transparent incorporation of the fault tolerance mechanisms into the application. The FTManager has two layers:

- The Communication Manager, which is responsible for the implementation of the communication algorithms, in order to support the dissemination of replica private values;
- The Replica Manager, which provides the necessary mechanisms for replica management, hiding its implementation from the application programmer.

Information regarding replication (replica configuration) is considered only at a final configuration phase. In such way, real-time applications can be programmed disregarding distribution and still use all the Ada powerful constructs.

Ada provides all the necessary mechanisms that are necessary to develop our architecture. Moreover, its strong compile-time type checking allows the easy detection of programming errors, extremely important in the fault-tolerant systems development.

5.1 Group Communication Hierarchy Example

Figure 2 presents the package hierarchy of the Multi-µ Group Communication support. The Group_Communication package provides the group abstraction for the higher layer, with a generic child implementing an agreement algorithm that can be instantiated by the higher layer, only if it is necessary, and only for the necessary data types.

The child package concept of Ada allows the functionality extension without changes to the father's code. Package SM_Algorithm is a child package of Group_Communication, providing extended capabilities without disrupting its father implementation. Other extensions can be provided, such as atomic broadcast or clock synchronisation.

The modularity that can be achieved with Ada hierarchical packages is presented by the use of a private child package of SM_Algorithm, responsible for the data handling, allowing its modification, without disturbing its ancestors. Furthermore, as it is private it is not known from other packages.

5.1.1 Generic Package Example

The generic package SM_Algorithm is independent of the data type that must be agreed upon, being instantiated for each necessary type. Furthermore, the choice procedure is also dependent of the type of data. In some situations data disagreement can be detected through inequality; however, averaging data may be necessary to make the data agreement.

```
generic

      type Data_Type is private;
      type Data_Array is array(Node_Id range <>)
             of Data_Type;
      with procedure Choose(Data: out Data_Type;
             Datas: Data_Array);

package Group_Communication.SM_Algorithm is

-- Package specification

end Group_Communication.SM_Algorithm;
```

5.1.2 Protected Types Example

The use of protected types gives rise to a much easier concurrency control. Mutual exclusion is provided in the access to the protected resource *Group*, preventing interaction between sending and receiving tasks.

```
protected type Group(Ident: Group_Id) is

      procedure Send(Msg: Message; Node:
             Node_Id);
      procedure Broadcast(Msg: Message);
      entry Received(Msg: Message);
      entry Receive(Msg: out Message);

private

      -- Group internal data

end Group;
```

Entries provide a mechanism that allows the use of a “barrier” to queue a task depending on some condition, shown in the Receive entry that a task only is allowed inside the protected object, when there is a message.

```
entry Receive(Msg: out Message) when
       Message_Has_Arrived is
begin
      -- Gives the available message
end Receive;
```

5.1.3 Timed Entry Call Example

Waiting for a message in an entry, a task can be blocked. The possible occurrence of a fault can make the loss of a message. The select statement provides, among several others, the possibility of timeout implementation. This feature is used to prevent a task eternal block waiting for a message that will never appear.

```
while Finished = False loop
      select
            Grp.Receive(Msg);
            -- Message Processing
      or
            delay until Timeout;
            -- Error Processing
      end select;
end loop;
```

These and other features of the language make real-time programming easier, but at the some time powerful. That’s why we believe that Ada is a suitable language to real-time systems, allowing the full potential of a concurrent language, but also with a strong software engineering approach.

6. Conclusions

This paper addressed Ada and Java (Real-Time Java) capabilities to support Real-Time systems programming, 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, as it is a C/C++ based language, it is our opinion that Real-Time Java will always less predictable than Ada, which will remain as a more interesting language for both hard real-time and dependable applications.

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, pp95-107

[2] Gosling J., McGilton H., “The Java Language Environment: A White Paper”, Sun Microsystems, May 1996

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

[4] Uckun S., Gasperoni F., “Real-Time Java Requirements”, submitted to IEEE Spectrum, available at: *http://www.sdct.itl.nist.gov/~carnahan/real-time*

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

[6] ISO, Information technology - Programming languages - Ada, “Ada Reference Manual”, ISO/IEC 8652, 1995.

[7] Pinho L. and Vasques F., “Multi-µ: An Ada 95 Based Architecture for Fault Tolerance Support of Real-Time Systems”, to appear at ACM SIGAda’98, Washington D.C., USA, November 8-12, 1998

[8] RTEMS/C Applications User’s Guide. On-Line Applications Research Corporation (Sep. 1997). http://www.oarcorp.com.

[9] Schonberg, E. and Banner, B. The GNAT project: a GNU-Ada 9X compiler. In Proceedings of Tri’Ada’94 (Baltimore, USA, Nov. 1994), ACM Press, pp. 48-57.

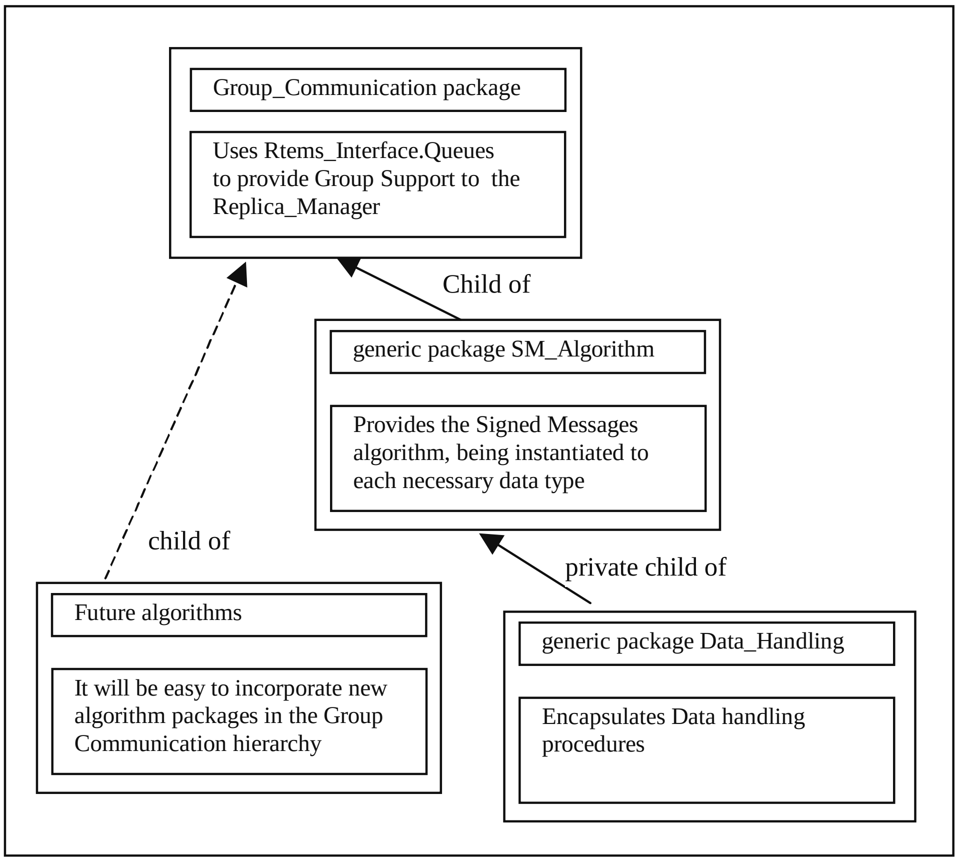
Group_Communication package
Uses Rtems_Interface.Queues to provide Group Support to the Replica_Manager
Child of
generic package SM_Algorithm
Provides the Signed Messages algorithm, being instantiated to each necessary data type
child of
private child of
Future algorithms
It will be easy to incorporate new algorithm packages in the Group Communication hierarchy
generic package Data_Handling
Encapsulates Data handling procedures

Fig. 2 - Group_Communication Hierarchy