Ada 2005 for Mission-Critical Systems

José F. Ruiz

AdaCore
8 rue de Milan
75009 Paris, France
Phone: +33 (0)1 49 70 67 16
Fax: +33 (0)1 49 70 05 52

ruiz@adacore.com

Abstract

For the development of mission-critical software, the choice of programming language makes a significant difference in meeting the requirements of exacting safety standards and, ultimately, high-reliability applications. Ada has a long history of success in the safety-critical domain, with features such as strong typing, that help early error detection, and well-defined semantics. The language has evolved based on user experience, and the forthcoming Ada 2005 standard includes a number of enhancements that will be of particular benefit to developers of high-integrity real-time systems. Relevant features include support for run-time profiles, flexible task-dispatching policies, execution-time clocks and timers, and a unification of concurrency and object-oriented features.

1 Introduction

Ada [9] is the language of choice for many critical systems due to its careful design and the existence of clear guidelines for building high integrity systems [10].

The Ada language was first introduced in 1983. Used primarily for large-scale safety and security critical projects (embedded systems in particular) where reliability and efficiency are essential, Ada experienced its last major revision in 1995, making it the first internationally standardized object-oriented language. The latest revision (Ada 2005) has been enhanced to better address the needs of the real-time and high-integrity communities. This new standard introduces new restrictions and configurations that can be used to define highly efficient, simple, and predictable run-time profiles. Among others, this language revision will standardize the Ravenscar profile, new scheduling policies, and will include execution time clocks and timers. Flexible object-oriented features are also supported without compromising performance or safety.

The following sections will describe the advantages of using Ada for developing mission-critical software, paying special attention to the new features that will be available in the forthcoming Ada 2005 standard.

2 Subsetting the language

Ada 2005 may be regarded not a single language but rather a family of languages. In practice, in writing mission-critical software, one does not want to use the full facilities of a complex language, since excessive complexity is an enemy of reliability. Instead a reduced subset is chosen that can be supported by a compact run-time system with a reduced footprint. Another advantage of this subsetting is the reduced complexity which facilitates the generation of proofs of correctness, predictability, reliability, or coverage analysis, if needed.

All general-purpose languages use this subset approach, but two aspects make Ada unique. First, the notion of such subsets is built into the language standard, rather than being external to it (the latter is exemplified by the attempt to define a "safe C" subset such as MISRA C). Second, the specific features in the subset can be chosen by the application developer, thus providing a high degree of flexibility. Indeed, such flexibility is essential in practice, since the exact set of features in the subset depends on the analysis techniques that are expected to be used during the development process, which in turn depends on the level of criticality.

One of the most interesting subsets for high-integrity systems is the Ravenscar profile, a collection of concurrency features that are powerful enough for real-time programming but simple enough to make certification practical. Another notable example is SPARK [2] that includes Ada constructs regarded as essential for the construction of complex software, but removes all the features that may jeopardize the requirements of verifiability, bounded space and time, and minimal run-time system.

3 The Ravenscar profile

As the functionality and complexity of embedded software increases, more attention is being devoted to high level, abstract development methods. The Ada tasking model provides concurrency as a means of decoupling application activities, and hence making software easier to design and test [15].

The tasking model in Ada is extremely powerful, but it has always been recognized that, in the case of mission-critical systems, it is appropriate to choose a subset of the tasking facilities because accurate timing analysis is difficult to achieve. Advances in real-time systems timing analysis methods have paved the way to reliable tasking in Ada; accurate analysis of real-time behavior is possible given a careful choice of scheduling/dispatching method together with suitable restrictions on the interactions allowed between tasks.

One of the most important achievements of Ada 2005 is the standardization of the Ravenscar restricted tasking profile. This profile [4] defines a subset of the tasking features of Ada which is amenable to static analysis for high integrity system certification, and that can be supported by a small, reliable run-time system. This profile is founded on state-of-the-art, deterministic concurrency constructs that are adequate for constructing most types of real-time software [5]. Major benefits of this model are:

- Improved memory and execution time efficiency, by removing high overhead or complex features.
- Increased reliability and predictability, by removing non-deterministic and non analysable features.
- Reduced certification cost by removing complex features of the language, thus simplifying the generation of proof of predictability, reliability, and safety.

Constructions that are difficult to analyze, such as dynamic tasks and protected objects, task entries, dynamic priorities, select statements, asynchronous transfer of control, relative delays, or calendar clock, are forbidden.

The concurrency model promoted by the Ravenscar Profile is consistent with the use of tools that allow the static properties of programs to be verified. Potential verification techniques include information flow analysis, schedulability analysis, execution-order analysis and model checking. It defines a computation model similar to the one proposed by Vardanega [14], which is based on the HRT-HOOD method [6]. The profile allows implementing space on-board systems using the tasking facilities provided by Ada, restricted so as to ensure that the system can be analysed for accurate timing and safety requirements. Preliminary experience confirms the validity of this approach for on-board software development [13].

The Ravenscar tasking model is static, so the complete set of tasks and associated parameters (such as their stack sizes) are identified and defined at compile time, and hence the required data structures (task descriptors and stacks) can be statically created by the compiler as global data. Therefore, memory requirements can be determined at link time and the use of dynamic memory can be avoided.

Ada 2005 contains determinism and hazard mitigation issues relating to task activation and interrupt handler execution semantics, in response to certification concerns about potential race conditions that could occur due to tasks being activated and interrupt handlers being executed prior to completion of the library-level elaboration code. A new configuration pragma has been added for guaranteeing the atomicity of program elaboration, that is, no interrupts are delivered and task activations are deferred until the completion of all library-level elaboration code. This eliminates all hazards that relate to tasks and interrupt handlers accessing global data prior to it having been elaborated, without having to resort to potentially complex elaboration order control.

Another major hazard in high-integrity systems, tasks terminating silently, has been addressed in Ada 2005 with a new mechanism for setting user-defined handlers which are executed when tasks are about to terminate. These procedures are invoked when tasks are about to terminate (either normally, as a result of an unhandled exception, or due to abort), allowing controlled responses at run time and also logging these events for post-mortem analysis.

The Ravenscar profile is part of the Ada 2005 standard, so compiler vendors must implement it. The intention is that not only will they support it, but in appropriate environments (notably embedded environments), efficient implementations of the Ravenscar tasking model will also be supplied.

4 Scheduling and dispatching policies

An important area of increased flexibility in Ada 2005 is that of task dispatching policies. In Ada 95, the only predefined policy is fixed-priority preemptive scheduling, although other policies are permitted. Ada 2005 provides further pragmas, policies, and packages which facilitate many different mechanisms such as non-preemption within priorities, round robin using timeslicing, and Earliest Deadline First (EDF) policy. Moreover, it is possible to mix different policies according to priority levels within a partition.

Time sharing the processor using round robin scheduling is adequate for non-real-time systems, and also in some soft real-time systems requiring a level of fairness. Many operating systems, including those compliant with the POSIX real-time scheduling model, support this scheduling policy that ensures that if there are multiple tasks at the same priority one of them will not monopolize the processor.

In order to reduce non-determinism and to increase the effectiveness of testing, non-preemptive execution is sometimes desirable [3]. The standard way of implementing many high-integrity applications is with a cyclic executive [1]. Using this technique a sequence of procedures is called within a defined time interval. Each procedure runs to completion and there is no concept of preemption. Data is passed from one procedure to another via shared variables and no synchronization constraints are needed, since the procedures never run concurrently. The major disadvantage with non-preemption is that it will usually (although not always) lead to reduced schedulability.

Ada 2005 supports the notion of deadlines (the most important concept in real-time systems) via a predefined task attribute. The deadline of a task is an indication of the urgency of the task. EDF scheduling allocate the processor to the task with the earliest deadline. EDF has the advantage that higher levels of resource utilization are possible, although it is less predictable, compared to fixed-priority scheduling, in case of overload situations.

5 Execution time monitoring and control

Monitoring and control execution time is important for many real-time systems. Ada 2005 provides an additional timing mechanism which allows for:

- monitoring execution time of individual tasks,
- defining and enabling timers and establishing a handler which is called by the run-time system when the execution time of the task reaches a given value, and
- defining a execution budget to be shared among several tasks, providing means whereby action can be taken when the budget expires.

Monitoring CPU usage of individual tasks can be used to detect at run time an excessive consumption of computational resources, which are usually caused by either software errors or errors made in the computation of worst-case execution times.

Schedulability analysis are based on the assumption that the execution time of each task can be accurately estimated. Measurement is always difficult, because, with effects like cache misses, pipelined and superscalar processor architectures, the execution time is highly unpredictable. Run-time monitoring of processor usage permits detecting and responding to wrong estimations in a controlled manner.

CPU clocks and timers are also a key requirement for implementing some modern real-time scheduling policies which need to perform scheduling actions when a certain amount of execution time has been consumed. Providing common CPU budgets to groups of tasks is the basic support for implementing aperiodic servers, such as sporadic servers and deferrable servers [12] in fixed priority systems, or the constant bandwidth server [8] in EDF-scheduled systems.

6 Timing events

Timing events allow for a handler to be executed at a future point in time in a efficient way, as it is a stand-alone timer which is execute directly in the context of the interrupt handler (it does not need a server task).

The use of timing events may reduce the number of tasks in a program, and hence reduce the overheads of context switching. It provides an effective solution for programming short time-triggered procedures, and for implementing some specific scheduling algorithms, such as those used for imprecise computation [11]. Imprecise computation increase the utilization and effectiveness of real-time applications by means of structuring tasks into two phases (one mandatory and one optional). Scheduling algorithms that try to maximize the likelihood that optional parts are completed typically require changing asynchronously the priority of a task, which can be implemented elegant and efficiently with timing events.

7 Object-oriented programming

Programmers writing high-integrity systems want to take advantage of the powerful notions of object-oriented programming, and work is being done in the direction of providing guidelines for certifying object-oriented applications [7]. Ada 2005 is ideally suited as the vehicle for exploiting what is safe in this area, while avoiding what is dangerous.

Type extension and inheritance are powerful and lightweight object-oriented mechanisms in Ada which are useful for embedded programing. Dynamic dispatching is also available, but there is a language defined restriction (*No_Dispatch*) that can be used for forbidding the use of dynamic dispatching, allowing for a more efficient and predictable execution. Ada 2005 offers also very fine-grained control over inheritance by allowing each operation to declare explicitly whether it is intended to inherit, and the compiler checks that the intention is met (this avoids accidentally confusing Initialize and Initialise for example, a well known hazard in object-oriented languages).

A conscious decision was made in the design of Ada 95 to not implement general multiple inheritance, because the complexities introduced to the language appeared to overwhelm the benefits. But more recently, the notion of interfaces (or roles) has been developed as an effective alternative that gives the power of interfacing to multiple abstractions without the additional complexity of full multiple inheritance. Java introduced the idea of interfaces, and Ada 2005 builds on the concept to create a new and powerful form of the interface abstraction, which also extends to the unique Ada notions of task and concurrent object, maintaining the important design principle that concurrency is a first class citizen.

8 Conclusions

Ada is a powerful and well-designed language, thoroughly reviewed as part of its standardization process, wich allows an effective and efficient use of high-level abstract development methods in mission-critical environments, without compromising performance or safety.

Safe tasking is promoted by the Ravenscar profile, which defines a deterministic and certifiable tasking subset, providing the high-level abstraction and expressive power needed for making software easy to design and test. Major hazards related to tasks terminating silently and potential race conditions at elaboration time have been addressed by new mechanisms added to Ada 2005.

The new language revision constitutes also the reference framework for reliable and efficient object-oriented programming, supporting powerful and flexible object-oriented features while avoiding those that jeopardize the behavior of the system.

References

[1] T.P. Baker and A. Shaw. The cyclic executive model and Ada. *Real-Time Systems*, 1(1), 1989.

[2] John Barnes. *High Integrity Software. The SPARK Approach to Safety and Security*. Addison Wesley, 2003.

[3] A. Burns. Defining new non-preemptive dispatching and locking policies for Ada. In D. Craeynest and A. Strohmeier, editors, *Reliable Software Technologies — Ada-Europe 2001*, number 2043 in Lecture Notes in Computer Science, pages 328–336. Springer-Verlag, 2001.

[4] Alan Burns. The Ravenscar profile. Technical report, University of York, 2002. Available at `http://www.cs.york.ac.uk/~burns/ravenscar.ps`.

[5] Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report YCS-2003-348, University of York, 2003. Available at `http://www.cs.york.ac.uk/ftpdir/reports/YCS-2003-348.pdf`.

[6] Alan Burns and Andy Wellings. *HRT-HOOD(TM): A Structured Design Method for Hard Real-Time Ada Systems*. North-Holland, Amsterdam, 1995.

[7] FAA. *Handbook for Object-Oriented Technology in Aviation (OOTiA)*, October 2004. Available at `http://www.faa.gov/certification/aircraft/av-info/software/OOT.htm`.

[8] T. M. Ghazalie and Theodore P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1):31–67, 1995.

[9] ISO. *Ada 95 Reference Manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995*, 1995. Available from Springer-Verlag, LNCS no. 1246.

[10] ISO/IEC/JTC1/SC22/WG9. *Guidance for the use of the Ada Programming Language in High Integrity Systems*, 2000. ISO/IEC TR 15942:2000.

[11] J. W. Liu, K. J. Lin, W. K. Shih, A. Chuang-Shi, J. Y. Chung, and W. Zhao. Algorithms for Scheduling Imprecise Computations. *IEEE Computer*, 24(5):58–68, May 1991.

[12] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1), 1989.

[13] T. Vardanega, G.Caspersen, and J.S. Pedersen. A case-study in the reuse of on-board embedded real-time software. In Michael González-Harbour and Juan A. de la Puente, editors, *Reliable Software Technologies — Ada-Europe'99*, number 1622 in LNCS, pages 425–436. Springer-Verlag, 1999.

[14] Tullio Vardanega. *Development of On-Board Embedded Real-Time Systems: An Engineering Approach*. PhD thesis, TU Delft, 1998. Also available as ESA STR-260.

[15] Tullio Vardanega and Jan van Katwijk. A software process for the construction of predictable on-board embedded real-time systems. *Software Practice and Experience*, 29(3):1–32, 1999.