

Ada 2005 for deeply embedded systems

By Jose Ruiz

For more than 20 years Ada has been used successfully to build embedded real-time systems with demanding requirements for reliability, safety, and performance. During this time 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 embedded systems.

The Ada 2005 standard features support for run-time profiles, flexible task-dispatching policies, execution-time clocks and timers, and a unification of concurrency and object-oriented features. Among these new facilities, one of the most significant is the standardization of the Ravenscar tasking profile. This subset of Ada tasking features embodies a deterministic concurrency model inherently amenable to static analysis and implementable by a small, reliable, and extremely efficient run-time library. The Ravenscar profile brings modern software engineering techniques to deeply embedded systems with limited resources by providing the abstraction and expressive power that help make software easier to design and test, but without compromising performance or safety.

Ada was designed with embedded applications in mind from the start. For example, the use of representation clauses, which have been extended and made more powerful in Ada 2005, allows close mapping of data structures to the hardware (the desired location, size, alignment, and layout of data in physical memory can be specified). The use of Ada high-level constructions facilitates development, code reviews, and analysis, and Ada's strong typing allows for compile-time verification that helps early detection of errors in the development process. Additionally, many embedded applications require high reliability or are safety critical, which allows a language designed for maximum safety to really shine. More than 99 percent of the aviation software in the Boeing 777 is in Ada.

Subsetting the language

Ada 2005 may be regarded not as a single language, but rather a family of languages. In practice, when writing embedded software, one does not want to use the full facilities of a complex language since resources are scarce. Instead, software writers can choose a reduced subset 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.

The notion of Ada subsets is built into the language standard, rather than being external to it (such as the MISRA C subset). Additionally, the application developer can choose the specific features in the subset, thus providing a high degree of flexibility. Any attempt to use excluded features will be flagged as a compile-time error, and importantly, the implementation can exclude from the executable any run-time support for what is forbidden, thus reducing the memory footprint of the application.

The Ravenscar tasking profile

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

The tasking model in Ada is extremely powerful, but it may be considered relatively complex, resource consuming, and difficult to analyze. The Ravenscar profile is a subset of Ada tasking that provides the basis for the implementation of simple, deterministic, and time-analyzable applications. This subset is amenable to static analysis for high integrity system certification, and can be supported by a small, reliable, and efficient run-time system.

The Ravenscar profile is founded on state-of-the-art, deterministic concurrency constructs adequate for constructing most types of real-time software. 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 nondeterministic and non-analyzable features
- Reduced certification cost by removing complex features of the language, thus simplifying the generation of proof of predictability, reliability, and safety

The profile is based on a computation model, which includes the following features:

- A single processor
- A fixed number of tasks
- A single invocation event per task (either time-triggered or event-triggered tasks)
- Task interaction only by means of shared data (protected objects) with mutually exclusive access

The tasking model defined by the profile includes tasks and protected types and objects at the library level, a maximum of one protected entry with a simple Boolean barrier for synchronization, a real-time clock, absolute delays, preemptive priority scheduling with ceiling locking access to protected objects, and protected procedure interrupt handlers, as well as some other features, which allow the development of embedded real-time systems.

Constructs 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. Ravenscar's simplicity allows memory usage and execution to be efficient and deterministic.

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; thus the required data structures (task descriptors and stacks) can be created statically by the compiler as global data. Therefore, memory requirements can be determined at link time, and the use of dynamic memory can be avoided.

The Ravenscar profile, part of the Ada 2005 standard, helps achieve source code portability. The intention is that not only will Ravenscar be implemented, but in appropriate environments (notably embedded environments), efficient implementations of the Ravenscar tasking model will also be supplied.

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 nonpreemption within priorities, round robin using timeslicing, and Earliest Deadline First (EDF). Moreover, it is possible to mix different policies according to priority levels within a partition.

Execution time monitoring and control

Monitoring and controlling 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 that is called by the run-time system when the execution time of the task reaches a given value
- Defining an 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 at run time to detect an excessive consumption of computational resources, which usually are caused by either software errors or errors made in the computation of worst-case execution times.

Schedulability analysis is based on the assumption that the execution time of each task can be estimated accurately. Measurement is always difficult, because with effects like cache misses and 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 key requirements for implementing some modern real-time scheduling policies, which must perform scheduling actions when a certain amount of execution time has been consumed. Providing common CPU budgets to groups of tasks offers the basic support for implementing aperiodic servers, such as sporadic servers and deferrable servers in fixed priority systems, or the constant bandwidth server in EDF-scheduled systems.

Timing events

Timing events allow for a handler to be executed at a future point in time in an efficient way, as it is a standalone timer that is executed 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 the overhead of context switching. It provides an effective solution for programming short time-triggered procedures and implementing some specific scheduling algorithms, such as those used for imprecise computation. Imprecise computation increases 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 elegantly and efficiently with timing events.

Object-oriented programming

Developers of embedded software often want to take advantage of the power of object-oriented programming. Given Ada's emphasis on flexibility and reliability, Ada 2005 directly addresses the use of object-oriented methods in a configurable and safe way.

Ada's type extension and inheritance are powerful and lightweight object-oriented mechanisms useful for embedded programming. Dynamic dispatching is also available, but a language-defined restriction (*No_Dispatch*) can be used for forbidding its use, allowing for more efficient and predictable execution. Ada 2005 offers very fine-grained control over inheritance by allowing each operation to declare explicitly if 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.

Designers of Ada 95 made a conscious decision to not implement general multiple inheritance because they thought its complexities overwhelmed 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 proved the value of the interface concept, and Ada 2005 builds on this notion 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.

Ada 2005, the key to efficiency and reliability

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

Reliable and efficient tasking is promoted by the Ravenscar profile, which defines a deterministic and certifiable tasking sub-set, providing the high-level abstraction and expressive power needed for making software easy to design and test.

Ada 2005 also supports reliable and efficient object-oriented programming by providing a high level of configurability in how to use dynamic dispatching, and by smoothly combining the concurrency and object-orientation features. **ECD**

Jose F. Ruiz is a senior software engineer at AdaCore. He received his PhD degree for his work in the field of real-time and multimedia systems, including scheduling policies and resource management in real-time operating systems. He has been working on embedded real-time Ada for more than 15 years, and has authored/coauthored more than 20 papers in that area.



For more information, contact Jose at:

AdaCore

104 Fifth Avenue, 15th floor

New York, NY 10011

Tel: 212-620-7300 • Fax: 212-807-0162

E-mail: ruiz@adacore.com

Website: www.adacore.com

The Ada tasking model provides concurrency as a means of decoupling application activities, hence making software easier to design and test.