TECHNICAL REPORT

ISO/IEC TR 24718

First edition
2005-02-15

Information technology — Programming languages — Guide for the use of the Ada Ravenscar Profile in high integrity systems

Technologies de l'information — Langages de programmation — Guide pour l'usage de «Ada Ravenscar Profile» dans les systèmes de haute intégrité

Reference number
ISO/IEC TR 24718:2005(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 24718, which is a Technical Report of type 3, was prepared by the University of York for the British Standards Institution (BSI) as guidelines published in 2003, and was adopted (without modifications) by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Introduction

The use of Ada has proven to be of great value within high integrity and real-time applications, albeit via language subsets of deterministic constructs, to ensure full analysability of the code. Such subsets have been defined for Ada 83, but these have excluded tasking on the grounds of its non-determinism and inefficiency. Advances in the area of schedulability analysis currently allow hard deadlines to be checked, even in the presence of a run-time system that enforces preemptive task scheduling based on multiple priorities. This valuable research work has been mapped onto a number of new Ada constructs and rules that have been incorporated into the Real-Time Annex of the Ada language standard. This has opened the way for these tasking constructs to be used in high integrity subsets whilst retaining the core elements of predictability and reliability.

The Ravenscar Profile is a subset of the tasking model, restricted to meet the real-time community requirements for determinism, schedulability analysis and memory-boundedness, as well as being suitable for mapping to a small and efficient run-time system that supports task synchronization and communication, and which could be certifiable to the highest integrity levels. 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. These techniques allow analysis of a system to be performed throughout its development life cycle, thus avoiding the common problem of finding only during system integration and testing that the design fails to meet its non-functional requirements.

The Ravenscar Profile has been designed such that the restricted form of tasking that it defines can be used even for software that needs to be verified to the very highest integrity levels. The aim of this guide is to give a complete description of the motivations behind the Profile, to show how conformant programs can be analysed and to give examples of usage.

Information technology — Programming languages — Guide for the use of the Ada Ravenscar Profile in high integrity systems

1 Scope

This Technical Report provides a description of the motivations behind the Ravenscar Profile, to show how Ada programs using the profile can be analysed, and gives examples of usage.

2 Recommendations

The technical recommendations are those made in the following publication (reproduced on the following pages), which is adopted as a Technical Report:

Guide for the use of the Ada Ravenscar Profile in high integrity systems, Alan Burns, Brian Dobbing, and Tullio Vardanega, University of York Technical Report YCS-2003-348, January 2003.

For the purposes of international standardization, the modifications outlined below shall apply to the specific clause and paragraphs of the University of York publication.

Page i to ii (of the University of York publication)

This is information relevant to the University of York publication only.

Page 73

Clause 9

Substitute the following for the corresponding reference

[GA] ISO/IEC TR 15942:2000, *Information technology — Programming languages — Guide for the use of the Ada programming language in high integrity systems*

[RM] ISO/IEC 8652, *Information technology — Programming languages — Ada*

3 Revision of the University of York publication

It has been agreed with the University of York that ISO/IEC JTC 1/SC 22 will be consulted in the event of any revision or amendment of this University of York publication. To this end, the British Standards Institution (BSI) will act as a liaison body between the University of York and ISO/IEC.

Guide for the use of the Ada Ravenscar Profile in high integrity systems

Alan Burns, Brian Dobbing and Tullio Vardanega

University of York Technical Report YCS-2003-348
January 2003

Blank page

Contents

1 Introduction

There is increasing recognition that the software components of critical real-time applications must be provably predictable. This is particularly so for a hard real-time system, in which the failure of a component of the system to meet its timing deadline can result in an unacceptable failure of the whole system. The choice of a suitable design and development method, in conjunction with supporting tools that enable the real-time performance of a system to be analysed and simulated, can lead to a high level of confidence that the final system meets its real-time constraints.

Traditional methods used for the design and development of complex applications, which concentrate primarily on functionality, are increasingly inadequate for hard real-time systems. This is because non-functional requirements such as dependability (e.g. safety and reliability), timeliness, memory usage and dynamic change management are left until too late in the development cycle.

The traditional approach to formal verification and certification of critical real-time systems has been to dispense entirely with separate processes, each with their own independent thread of control, and to use a *cyclic executive* that calls a series of procedures in a fully deterministic manner. Such a system becomes easy to analyse, but is difficult to design for systems of more than moderate complexity, inflexible to change, and not well suited to applications where sporadic activity may occur and where error recovery is important. Moreover, it can lead to poor software engineering if small procedures have to be artificially constructed to fit the cyclic schedule.

The use of Ada has proven to be of great value within high integrity and real-time applications, albeit via language subsets of deterministic constructs, to ensure full analysability of the code. Such subsets have been defined for Ada 83, but these have excluded tasking on the grounds of its non-determinism and inefficiency. Advances in the area of schedulability analysis currently allow hard deadlines to be checked, even in the presence of a run-time system that enforces preemptive task scheduling based on multiple priorities. This valuable research work has been mapped onto a number of new Ada constructs and rules that have been incorporated into the Real-Time Annex of the Ada language standard [RM D]. This has opened the way for these tasking constructs to be used in high integrity subsets whilst retaining the core elements of predictability and reliability.

The Ravenscar Profile is a subset of the tasking model, restricted to meet the real-time community requirements for determinism, schedulability analysis and memory-boundedness, as well as being suitable for mapping to a small and efficient run-time system that supports task synchronization and communication, and which could be certifiable to the highest integrity levels. 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. These techniques allow analysis of a system to be performed throughout its development life cycle, thus avoiding the common problem of finding only during system integration and testing that the design fails to meet its non-functional requirements.

It is important to note that the Ravenscar Profile is silent on the non-tasking (i.e. sequential) aspects of the language. For example it does not dictate how exceptions should, or should not, be used. For any particular application, it is likely that constraints on the sequential part of the language will be required. These may be due to other forms of *static analysis* to be applied to the code, or to enable *worst-case execution time* information to be derived for the sequential code. The reader is referred to the ISO Technical Report, Guide for the Use of Ada Programming

Language in High Integrity Systems [GA] for a detailed discussion on all aspects of static analysis of sequential Ada.

The Ravenscar Profile has been designed such that the restricted form of tasking that it defines can be used even for software that needs to be verified to the very highest integrity levels. The Profile has already been included in the ISO technical report [GA] referenced above. The aim of this guide is to give a complete description of the motivations behind the Profile, to show how conformant programs can be analysed and to give examples of usage.

Structure of the Guide

The report is organized as follows. The motivation for the development of the Ravenscar Profile is given in the next chapter. Chapter 3 includes the definition of the profile as agreed by *WG9*; the definition is included here for convenience, but this report is not the definitive statement of the profile. In Chapter 4, the rationale for each aspect of the profile is described. Examples of usage are then provided in Chapter 5. The need for verification is an important design goal for Ravenscar and Chapter 1 reviews the verification approach appropriate to Ravenscar programs. Finally in Chapter 7 an extended example is given. Definitions and references are included at the end of the report.

Readership

This report is aimed at a broad audience, including application programmers, implementers of run-time systems, those responsible for defining company/project guidelines, and academics. Familiarity with the Ada language is assumed.

Conventions

This report uses the *italics* face to flag the first occurrence of terms that have a defining entry in Chapter 8. For all Ada-related terms the report follows the language reference manual [RM] style: it uses the Arial font where there is a reference to defined syntax entities (e.g. delay_relative_statement). For all other names (e.g. Ada.Calendar) it uses normal text font, as do language keywords in the text except that they are in **bold** face.

2 Motivation for the Ravenscar Profile

Before describing the Ravenscar Profile in detail, we will explain in this chapter some of the reasoning behind its features. These primarily come from the need to be able to verify concurrent real-time programs, and to have these programs implemented reliably and efficiently.

In this chapter we look mainly at scheduling theory, as this is the main driver for the definition of the restrictions of the Profile. In addition there is a section that summarizes other program verification techniques that can be used with the Profile.

2.1 Scheduling Theory

Recent research in scheduling theory has found that accurate analysis of real-time behaviour is possible given a careful choice of scheduling/dispatching method together with suitable restrictions on the interactions allowed between tasks. An example of a scheduling method is *preemptive fixed priority scheduling*. Example analysis schemes are *Rate Monotonic Analysis* (*RMA*) [1] and *Response Time Analysis* (*RTA*) [2].

Priority-based preemptive scheduling is usually used with a *Priority Ceiling Protocol* (*PCP*) to avoid unbounded priority inversion and deadlocks. It provides a model suitable for the analysis of concurrent real-time systems. The approach supports *cyclic* and *sporadic* activities, the idea of *hard*, *soft*, *firm*, and *non-critical* components, and controlled inter-process communication and synchronization. It is also scalable to programs for distributed systems.

Tool support exists for RMA and RTA, and for the static simulation of concurrent real-time programs. The primary aim of analysing the real-time behaviour of a system is to determine whether it can be scheduled in such a way that it is guaranteed to meet its timing constraints. Whether the timing constraints are appropriate for meeting the requirements of the application is not an issue for scheduling analysis. Such verification requires a more formal model of the program and the application of techniques such as model checking – see section 2.4.

2.1.1 Tasks Characteristics

The various tasks in an application will each have timing constraints. For *critical tasks* these are normally defined in terms of *deadlines*. The deadline is the maximum time within which a task must complete its operation in response to an event.

Each task is classified into one of the following four basic levels of criticality according to the importance of meeting its deadline:

- Hard
  A *hard deadline task* is one that must meet its deadline. The failure of such a task to meet its deadline may result in an unacceptable failure at the system level.
- Firm
  A *firm deadline task* is one that must meet its deadline under “average” or “normal” conditions. An occasional missed deadline can be tolerated without causing system failure (but may result in degraded system performance). There is no value, and thus there is a system-level degradation of service, in completing a firm task after its deadline.

- Soft
  A *soft deadline task* is also one that must meet its deadline under "average" or "normal" conditions. An occasional missed deadline can be tolerated without causing system failure (but may result in degraded system performance). There is value in completing a soft task even if it has missed its deadline.
- Non-critical
  A *non-critical task* has no strict deadline. Such a task is typically a background task that performs activities such as system logging. Failure of a non-critical task does not endanger the performance of the system.

2.1.2 Scheduling Model

At any moment in time, some tasks may be *ready* to run (meaning that they are able to execute instructions if processor time is made available). Others are *suspended* (meaning they cannot execute until some event occurs) or *blocked* (meaning that they await access to a shared resource that is currently exclusively owned by another task). Suspended tasks may become ready synchronously (as a result of an action taken by a currently running task) or asynchronously (as a result of an external event, such as an interrupt or timeout, that is not directly stimulated by the current task).

With priority-based preemptive scheduling on a single processor, a priority is assigned to each task and the scheduler ensures that the highest priority ready task is always executing. If a task with a priority higher than the currently running task becomes ready, the scheduler performs a *context switch*, as soon as it can, to enable the higher-priority task to resume execution. The term "preemptive" indicates that this can occur because of an asynchronous event (i.e. one that is not caused by the running task).

Tasks will normally be required to interact as a result of contention for shared resources, exchange of data, and the need to synchronize their activities. Uncontrolled use of such interactions can lead to a number of problems:

- Unbounded *Priority Inversion* / Blocking
  where a high-priority task is *blocked* awaiting a resource in use by a low-priority task; as a result, ready tasks of intermediate priority may hold up the high priority task for an unbounded amount of time since they will run in preference to the low priority task that has locked the resource.
- *Deadlock*
  where a group of tasks (possibly the whole system) block each other permanently due to circularities in the ownership of and the contention for shared resources.
- *Livelock*
  where several tasks (possibly comprising the whole system) remain ready to run, and do indeed execute, but which fail to make progress due to circular data dependencies between the tasks that can never be broken.
- Missed *Deadline*
  where a task fails to complete its response before its deadline has expired due to factors such as system overload, excessive preemption, excessive blocking, deadlocks, livelocks or CPU overrun.

The restricted scheduling model that is defined by the Ravenscar Profile is designed to minimize the upper bound on blocking time, to prevent deadlocks, and (via tool support) to verify that there is sufficient processing power available to ensure that all critical tasks meet their deadlines.

In this model, tasks do not interact directly, but instead interact via shared resources known as *protected objects*. Each protected object typically provides either a resource access control function (including a repository for the private data to manage and implement the resource), or a synchronization function, or a combination of both.

A protected object that is used for resource access control requires a mutual exclusion facility, commonly known as a *monitor* or *critical region*, where at most one task at a time can have access to the object. During the period that a task has access to the object, it must not perform any operation that could result in it becoming suspended. Ada directly supports protected objects and disallows internal suspension within these objects.

A protected object that is used for synchronization provides a signalling facility, whereby tasks can signal and/or wait on events. In the Profile definition, the use of protected objects for synchronization by the critical tasks is constrained so that at most one task can wait on each protected object. A simplified version of wait/signal is also provided in the Profile via the Ada Real-Time Annex functionality known as *suspension objects* [RM D.10]. These can be used in preference to the protected object approach for simple resumption of a suspended task, whereas the protected object approach should be used when more complex resumption semantics are required, for example including deterministic (*race-condition*-free) exchange of data between signaller and waiter tasks.

The Profile definition assures absence of deadlocks by requiring use of an appropriate locking policy. This policy requires a *ceiling priority* to be assigned to each protected object that is no lower than the highest priority of all its calling tasks, and results in the raising of the priority of the task that is using the protected object to this ceiling priority value. In addition to absence of deadlocks, this policy also allows an almost optimal time bound on the worst case blocking time to be computed for use within the schedulability analysis, thereby eliminating the unbounded priority inversion problem. This time bound is calculated as the maximum time that the object is in use by lower-priority tasks. Therefore, the smaller the worst-case time bound for this blocking period, the greater the likelihood that the task set will be schedulable.

The use of priority-based preemptive dispatching defines a mechanism for scheduling. The scheduling policy is defined by the mapping of tasks to priority values. Many different schemes exist for different temporal characteristics of the tasks and other factors such as criticality. What most of these schemes require is an adequate range of distinct priority values. Ada and the Ravenscar Profile ensure this.

2.2 Mapping Ada to the Scheduling Model

The analysis of an Ada application that makes unrestricted use of Ada run-time features including tasking rendezvous, select statements and abort is not currently feasible. In addition, the non-deterministic and potentially unbounded behaviour of many tasking and other run-time calls may make it impossible to provide the upper bounds on execution time that are required for schedulability analysis and simulation. Thus Ada coding style rules and subset restrictions must be followed to ensure that all code within critical tasks is statically time-bounded, and that the execution of the tasks can be defined in terms of response times, deadlines, cycle times, and blocking times due to contention for shared resources.

The application must be decomposed into a number of separate tasks, each with a single thread of control, with all interaction between these tasks identified. Each task has a single primary invocation event. The tasks are categorized as *time-triggered* (meaning that they execute in