An Introduction to the Ravenscar Profile

Tullio Vardanega
University of Padua, Italy
member of the HRG

SIGAda 2003

Outline of Contents

- Background
- The Ravenscar Profile
- History in Brief
- Understanding the Profile
- Examples of Use
- Conclusions

Background *1*

- Ada has earned a prominent role in the development of critical real-time systems
  - Via support to good software engineering practice
  - Via language subsets of deterministic constructs
- Ada 83 was designed to provide language-level support to concurrency and real-time
  - A brave decision, with an unintended deferred impact

Background

- Real-time systems are concurrent in nature
- Yet, traditional design has dispensed with multiple threads of control entirely
  - Obviously a problem of the concurrency model, which made design and verification difficult
  - Ada tasks were consequently viewed as
    - Non-deterministic
    - Inefficient
    - Heavy weight

Background

- Advances in the scheduling analysis theory
  - Have lifted the ban on preemptive priority based dispatching
  - Have been mapped on new concurrency models, well equipped for predictability
- Ada 95 has taken full advantage of this
  - New tasking constructs and rules incorporated in the Real-Time Annex
  - Now a superb tasking model

The Ravenscar Profile 1

- A subset of the Ada 95 tasking model
  - Silent on the sequential part of the language
- Restrictions designed to meet the real-time community requirements for
  - Determinism
  - Schedulability analysis
  - Memory-boundedness
  - Execution efficiency and small footprint
  - Suitability for certification

The Ravenscar Profile

- Championed by the IRTAW group on behalf of the real-time community
- Work started on the basis of Ada 95
  - Definition of a tasking model that warrant static scheduling analysis using standard dispatching policies and well-known analysis schemes
    - Fixed-priority preemptive dispatching
    - Response time analysis

The Ravenscar Profile

- Not the sole strand of work carried out by the IRTAW group
  - Very active in the definition and promotion of enhancements to the full Ada tasking model
- The Ravenscar Profile is a powerful catalyst to the promotion of simple and effective language-level concurrency
  - Crucial to critical applications
  - One end of the road to greater expressive power

History in Brief *1*

- April 1997, 8th IRTAW
  - First definition of the profile, named after the small Yorkshire village that hosted the event
- March 1999, 9th IRTAW
  - Definition reaffirmed, clarified and published in Ada Letters
  - Suspension objects allowed in
- September 2000, 10th IRTAW
  - Very positive use and implementation reports
  - Profile definition submitted to the ARG

History in Brief

- April 2002, 11th IRTAW
  - ARG definition of the Profile is agreed upon
  - Confirmed the need for
    - FIFO_Within_Priorities dispatching policy
    - Ceiling_Locking locking policy
- December 2002, WG9
  - 2 approved AIs formally define the Profile
    - AI-00305 new pragmas and additional restrictions (D.7 and H.5)
    - AI-00249 configuration pragma Profile with Ravenscar runtime profile argument (D.13)

History in Brief

- December 2000, WG9
  - The HRG charged to produce a rationale for the Profile
- January 2003, HRG
  - "*Guide for the use of the Ada Ravenscar Profile in high integrity systems*" published by the University of York as public technical report (ftp://ftp.cs.york.ac.uk/reports/YCS-2003-348.pdf)
- December 2003, WG9
  - The HRG Guide to become an ISO TR

History in Brief

- January 2001, ESA
  - Ada Ravenscar Products Evaluation Programme
    - A 6-month sponsored initiative for European space industy to evaluate the expressive power of the Profile and the adequacy of the Ada Ravenscar Technology
    - 2 early-adopter products
      - Aonix ObjectAda/RAVEN
      - UPM GNAT/ORK
    - 8 small case studies covering a range of software components of typical space applications
- November 2001, ESA
  - Evaluators workshop
    - http://www.estec.esa.nl/wmwww/EME/Ravenscar_Evaluation/proceedings.htm

History in Brief

- Evaluators reported
  - Happy with improved expressive power over alternative custom solutions
    - Profile restrictions were found to be more liberal than self-imposed ones
  - Could meet all known application requirements
    - Need to understand how to express timeouts without dropping restrictions
  - Would like to see guidance material
    - Design and coding patterns

Understanding the Profile 1

- The Ravenscar Profile is an alternative mode of operation defined by the standard
  - `pragma Profile (Ravenscar)`
  - Equivalent to a set of configuration pragmas
- Any run-time profile is legally expressed as a collection of restrictions
- We shall first understand what it allows
  - Then we will move on to what it prohibits

Understanding the Profile 2

- The Profile allows
  - Task and protected types and objects at library level
    - Task type and protected type discriminants
    - Protected procedure as statically bound interrupt handler
    - Max 1 entry per protected object with max 1 task queued at any time - Barrier must be a single Boolean variable
    - Use of **`E'Count`** in protected entries bodies
  - Atomic and Volatile pragmas
  - **`delay_until`** statements
  - Synchronous task control
  - `Ada.Real_Time`

Understanding the Profile

The profile corresponds to

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
pragma Detect_Blocking;
pragma Restrictions (
                Max_Entry_Queue_Length => 1,
                Max_Protected_Entries => 1,
                Max_Task_Entries => 0,
                No_Abort_Statements,
                No_Asynchronous_Control,
                No_Calendar,
                No_Dynamic_Attachment,
                No_Dynamic_Priorities,
                No_Implicit_Heap_Allocations,
                No_Local_Protected_Objects,
                No_Protected_Type_Allocators,
                No_Relative_Delay,
                No_Requeue_Statements,
                No_Select_Statements,
                No_Task_Allocators,
                No_Task_Attributes_Package,
                No_Task_Hierarchy,
                No_Task_Termination,
                Simple_Barriers);
```

AI-00305

Understanding the Profile 4

- Static existence model
  - Restrictions ensure that the set of tasks and interrupts to be analysed is fixed and has static attributes after program elaboration
- Static synchronisation and communication model
  - No rendezvous for task synchronisation and communication
  - Local protected objects meaningless for intertask synchronisation and communication

Understanding the Profile

- Deterministic memory usage
  - No implicit dynamic memory allocation by the implementation
  - Use of standard or user-defined storage pool via explicit allocators allowed
    - User must have visibility or control over how the pool is managed
- Deterministic execution model
  - No task queues on protected entries
  - No > 1 barriers becoming open simultaneously
  - Simpler runtime code

Understanding the Profile

- Enforced runtime detection of potentially blocking operations within a protected operation
  - Stronger requirement on runtime → new H.5(1)
  - Only few cases left from 9.5.1(9-16)
    - Protected entry calls and **`delay_until`** statements
  - Detection at point of execution allowed as opposed to at point of call
  - Allows efficient and temporally deterministic implementation of **`Ceiling_Locking`** policy
    - Ceiling priority on uniprocessors needs no locks and causes no queueing

Examples of use


```
task type Cyclic (Pri            : System.Priority;
                  Cycle_Time : Positive) is
   pragma Priority (Pri);
end Cyclic;
task body Cyclic is
   Next_Period :          Ada.Real_Time.Time;
   Period      : constant Ada.Real_Time.Time_Span :=
      Ada.Real_Time.Microseconds (Cycle_Time);
   -- Other declarations
begin
   -- Initialization code
   Next_Period := Ada.Real_Time.Clock + Period;
   loop -- wait one whole period before executing
      delay until Next_Period;
      -- Non-suspending periodic response code
      -- May include calls to protected procedures
      Next_Period := Next_Period + Period;
   end loop;
end Cyclic;
-- 2 task objects of this type
A_Cyclic_Task       : Cyclic (20,200);
Another_Cyclic_Task : Cyclic (15,100);
```

Examples of use

```
-- A suspension object SO is declared in a visible library unit
-- and is set to True in another (releasing) task
task type Sporadic (Pri : System.Priority) is
   pragma Priority (Pri);
end Sporadic;

task body Sporadic is
   -- Declarations
begin
   -- Initialization code
   loop
      Ada.Synchronous_Task_Control.Suspend_Until_True (SO);
      -- Non-suspending sporadic response code
   end loop;
end Sporadic;

An_Event_Triggered_Task : Sporadic (17);
```

Examples of use


```
protected type Event (Ceiling : System.Priority) is
   entry     Wait   (D :    out Data);
   procedure Signal (D : in     Data);
private -- Ceiling priority defined for each object
   pragma Priority (Ceiling);
   Current   : Data; -- Event data declaration
   Signalled : Boolean := False;
end Event;

protected body Event is
   entry Wait (D : out Data) when Signalled is
   begin
      D := Current;
      Signalled := False;
   end Wait;
   procedure Signal (D : in Data) is
   begin
      Current   := D;
      Signalled := True;
   end Signal;
end Event;
```

Examples of use


```
Event_Object : Event (15);

task Event_Handler is
   pragma Priority (14); -- must be not greater than 15
end Event_Handler;

task body Event_Handler is
   -- Declarations, including D of type Data
begin
   -- Initialization code
   loop
      Event_Object.Wait (D);
      -- Non-suspending event handling code
   end loop;
end Event_Handler;
```

Examples of use 5

```
select
   PO.Call;
   Timeout := False;
or
   delay until Some_Time;
   Timeout := True;
end select;
```

```
protected PO is
   entry Call (Timeout : out Boolean);
   procedure Release_Call;
   procedure Time_Out;
private
   Timed_Out : Boolean := False;
   Release   : Boolean := False;
end PO;
```

A standard timed entry call in full Ada can be expressed in Ravenscar by a composite structure , which:

- extends PO to return the timeout value to the user
- models the user as an event-triggered task
- employs an extra task to deliver the timeout event

Examples of use


```
protected body PO is
   procedure Time_Out is
   begin
      if Call'Count = 1 then
         Timed_Out := True;
         Release   := True;
      end if;
   end Time_Out;
   procedure Release_Call is
   begin
      Timed_Out := False;
      Release   := True;
   end Release_Call;
   entry Call (Timeout : out Boolean) when Release is
   begin
      Timeout := Timed_Out;
      Release := False;
      -- further non-suspending code if necessary
   end Call;
end PO;
```

The user task first arms the timeout task and then calls PO.Call

After receiving the timeout value the timeout task suspends itself until Some_Time and then calls PO.Time_Out

The partner task calls PO.Release_Call to complete the synchronisation

Conclusions

- Ada has a superb tasking model, which the current revision process will further improve
- Critical real-time applications need safe tasking
  - The Ravenscar Profile is a best-fit response to this need
  - An excellent vehicle to get more users into well-designed language-level concurrency
- The Ravenscar Profile cannot compete with the full language for expressive power