Ada Action Items

In subsequent issues depending upon the space availability ACM Ada Letters will publish Ada Action Items beginning with “Real Time AIs”. This issue publishes the first five Real Time AIs

AI 00249 - Ravenscar profile for high-integrity systems
AI 00297 - Timing events
AI 00298 - Non-Preemptive Dispatching
AI 00303 - Removal of library-level requirement for interrupt handler objects
AI 00305 - New pragma and additional restriction identifiers for real-time systems

AI 00249 - Ravenscar Profile for High-Integrity Systems

!standard D.13 (01) 05-02-25 AI95-00249/11

!class amendment 00-12-04
!status Amendment 200Y 02-10-23
!status WG9 Approved 02-12-13
!status ARG Approved 9-0-0 02-10-13
!status work item 00-12-04
!status received 00-12-04
!priority High
!difficulty Medium
!subject Ravenscar profile for high-integrity systems

!summary

A new pragma directly supports the Ravenscar Profile -- an execution-time profile suitable for use in High-Integrity and Safety-Critical applications.

!problem

The Ravenscar Profile is commonly used in Safety-Critical and High-Integrity applications. However, the profile is incompletely supported by the standard, requiring users to fall back on vendor-specific extensions.

!proposal

This amendment defines a pragma-based mechanism to allow applications to request use of the Ravenscar Profile. It adds a pragma to support the concept of a run-time profile. It also defines the run-time profile identifier "Ravenscar" and specifies its semantics.

The proposal assumes that additional Restriction identifiers and pragma Detect_Blocking have been defined (see AI-305).

A run-time profile is an alternative mode of operation that is defined by the standard. It is selected by inclusion of the configuration pragma Profile that applies to an active partition. The profile identifier "Ravenscar" selects the mode of operation to be the Ravenscar Profile.

!wording

Add new clauses D.13 and D.13.1

D.13 Run-time Profiles

This clause specifies a mechanism for defining run-time profiles.

Syntax
The form of a pragma Profile is as follows:
pragma Profile (profile_identifier {, profile_pragma_argument_association});

Legality Rules
The profile_identifier shall be the name of a run-time profile.
The semantics of any profile_pragma_argument_associations are defined by the run-time profile specified by the profile_identifier.

Static Semantics
A profile is equivalent to the set of configuration pragmas that is defined for each run-time profile.

Post-Compilation Rules
A pragma Profile is a configuration pragma. There may be more than one pragma Profile for a partition.

D.13.1 The Ravenscar Profile

This clause defines the Ravenscar profile.

Legality Rules
The profile_identifier Ravenscar names a run-time profile. For run-time profile Ravenscar, there shall be no profile_pragma_argument_associations.

Static Semantics
The run-time profile Ravenscar is equivalent to the following set of pragmas:

```
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);
```

NOTES
The effect of the Max_Entry_Queue_Length => 1 restriction applies only to protected entry queues due to the accompanying restriction of Max_Task_Entries => 0.

!example

To use the Ravenscar profile for a partition, we need to compile a pragma Profile before compiling any units:

```
    pragma Profile (Ravenscar);
```

!discussion

To give Ravenscar prominence it is given its own section within Annex D.
Moving the definition of pragma Profile to 13.12 would cause considerable disruption to that section.

Other profiles may be defined by an implementation, for example to define the SPARK subset, or to define variations of the Ravenscar profile such as Ravenscar_Non_Preemptive.

The restrictions forcing the maximum length of an entry queue to be one and the maximum number of entries to be one are there for deterministic operation of entry queue servicing and to simplify the run-time implementation. The semantics are intended to be consistent with those for blocking on a suspension object via Suspend_Until_True.

Static attachment of interrupt handlers via pragma Attach_Handler is supported.

!corrigendum D.13 (01)

@dinsc

This clause specifies a mechanism for defining run-time profiles.

@i<@s8<Syntax>>

The form of a pragma Profile is as follows:@hr
@xindent<@b<pragma> Profile (@i<profile_>@fa<identifier>
{@i<profile_>@fa<pragma_argument_association>);>

@i<@s8<Legality Rules>>

The @i<profile_>@fa<identifier> shall be the name of a run-time profile.
The semantics of any @i<profile_>@fa<pragma_argument_association>s are defined by the run-time profile specified by the @i<profile_>@fa<identifier>.

@i<@s8<Static Semantics>>

A profile is equivalent to the set of configuration pragmas that is defined for each run-time profile.

@i<@s8<Post-Compilation Rules>>

A pragma Profile is a configuration pragma. There may be more than one pragma Profile for a partition.

!corrigendum D.13.1 (01)

@dinsc

This clause defines the Ravenscar profile.

@i<@s8<Legality Rules>>

The @i<profile_>@fa<identifier> Ravenscar names a run-time profile.
For run-time profile Ravenscar, there shall be no
@i<profile_>@fa<pragma_argument_association>s.

@i<@s8<Static Semantics>>

The run-time profile Ravenscar is equivalent to the following set of pragmas:

```
@xcode<@b<pragma> Task_Dispatching_Policy (FIFO_Within_Priorities);
@b<pragma> Locking_Policy (Ceiling_Locking);
@b<pragma> Detect_Blocking;
@b<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);>
```

@xindent<@s9<NOTES@hr
37 The effect of the Max_Entry_Queue_Length =@> 1 restriction applies only to protected entry queues due to the accompanying restriction of Max_Task_Entries =@> 0.>>

! ACATS test

An ACATS test should be created for this pragma.

!appendix

!topic Addition of support for Ravenscar Profile
!reference RM-D
!from Brian Dobbing 00-09-24, Alan Burns 00-11-17, Joyce Tokar 00-11-17
!keywords Ravenscar, high integrity
!discussion

This comment proposes the addition of support in the language standard for the de facto standard Ravenscar Profile. This profile is included in the ISO document "Guide for the use of the Ada programming language in high integrity systems".

The comment proposes addition of pragma Ravenscar to Annex D of the RM as follows.

Pragma Ravenscar defines an alternative mode of operation that consists of the following amendments to the dynamic semantics of the standard. [Note: I say "alternative" rather than "non-standard" as in section 1.5. This is because its inclusion in the standard makes it not non-standard.]

Static Semantics

    pragma Ravenscar;

Pragma Ravenscar is a configuration pragma that applies to an active partition.

Dynamic Semantics

1 Task Dispatching Policy

The default Task_Dispatching_Policy for the active partition is FIFO_Within_Priorities. An implementation may define alternative task dispatching policies.

2 Locking Policy

The default Locking_Policy for the active partition is Ceiling_Locking. An implementation may define alternative locking policies.

3 Restrictions pragmas

3.1 Standard Pragmas

The following pragma Restrictions identifiers defined in the standard apply to the alternative mode of operation defined by pragma Ravenscar:

    Max_Asynchronous_Select_Nesting => 0
    Max_Task_Entries => 0
    Max_Protected_Entries => 1
    No_Abort_Statements
    No_Asynchronous_Control
    No_Dynamic_Priorities
    No_Implicit_Heap_Allocations
    No_Task_Attributes
    No_Task_Allocators
    No_Task_Hierarchy

The pragma No_Task_Hierarchy imposes the constraint that all tasks must depend immediately on the Environment task as a result of all task objects being created by library level declarations. [Note: It is not clear to me whether the existing RM wording fully implies this. It is not sufficient to say that all tasks shall depend (directly or indirectly) on the Environment task since this presumably would include those declared inside library-level subprograms. What we want to restrict is having to support "masters" and "waiting for dependent tasks"].

3.2 Additional Pragmas

The following new pragma Restrictions identifiers are defined as applying to the alternative mode of operation defined by pragma Ravenscar: [Aside: Alan Burns has a more complete write-up of these pragmas than is presented here.]

    Max_Entry_Queue_Depth = 1

Max_Entry_Queue_Depth defines the maximum number of calls that are queued on a (protected) entry. Violation of this restriction results in the raising of Program_Error exception at the point of the call. [Note: These semantics are intended to be consistent with those for blocking on a suspension object via Suspend_Until_True.]

    No_Calendar

There are no semantic dependencies on package Ada.Calendar.

    No_Dynamic_Attachment

There is no call to any of the operations defined in package Ada.Interrupts (Is_Reserved, Is_Attached, Current_Handler, Attach_Handler, Exchange_Handler, Detach_Handler, Reference) [Note: Static attachment via pragma Attach_Handler is supported.] {Note: At IRTAW-10, we called this No_Dynamic_Interrupt_Handlers, but that doesn't really convey the correct meaning of the restriction, hence my alternative suggestion.]

No_Local_Protected_Objects

All protected objects are created via library-level declarations. [Note: This matches the additional semantics for No_Task_Hierarchy. If these additional semantics were considered to be unacceptable, we could introduce No_Local_Task_Objects to go with this one.]

No_Protected_Type_Allocators

There are no allocators for protected types or types containing protected type components. [Note: This is like No_Task_Allocators]

No_Relative_Delay

Delay_relative statements are not allowed.
[Note: Unfortunately, No_Delay in H.4 is too strong since we want to allow delay_until Ada.Real_Time.Time, but not relative delay (non-deterministic) or Ada.Calendar (too coarse).]

No_Requeue

Requeue statements are not allowed.

No_Select_Statements

Select_statements are not allowed.
[Note: This includes selective_accept (there are no task entries anyway), timed and conditional entry calls, and asynchronous_select (which is re-inforced by Max_Async_Select_Nesting = 0.]

No_Task_Termination

No_Task_Termination defines task termination (including for the Environment task) to be a bounded error condition. The possible effects are:

The task terminates as defined by the standard mode of operation.

Prior to termination as defined by the standard mode of operation, the task calls an implementation-defined subprogram. The mechanisms for specifying the subprogram to be called, and any restrictions on the operation of the subprogram, are implementation-defined.
[Note: This is defined so as to mitigate the hazard of "silent task death" in high integrity systems and to provide the opportunity for application-specific recovery action.]

Simple_Barrier_Variables

The Boolean expression in an entry barrier shall be the value of a Boolean component of the enclosing protected object.

4 Outstanding Issue -- Task_Activation

To satisfy the requirements of the Safety Critical and High-Integrity domains, there is a need to define the behavior of program elaboration to be atomic. That is to say, that no interrupts are delivered during this period of execution. In addition, task activation shall be deferred until the completion of all elaboration code.

A proposed approach to addressing this concern is to introduce an optional argument on the pragma Ravenscar such that the activation of library level tasks may be specified. This would have the effect of the modifying the syntax and semantics for pragma Ravenscar as follows:

Static Semantics

pragma Ravenscar[(Task_Activation => Deferred | Standard)];

Dynamic Semantics

4.1 Task_Activation => Deferred

With the Deferred value for the Task_Activation argument, all task activation and all interrupt handler attachment for library-level tasks and interrupt handlers are deferred. The deferred task activation and handler attachment occurs immediately after the "begin" of the Environment task. At this point, the Environment task is suspended until all deferred task activation and handler attachment is complete. In this mode of operation, it is a bounded error for the Environment task to execute a call to Ada.Synchronous_Task_Control.Suspend_Until_True or to execute a protected entry call that is blocking during its declarative part. Program_Error may be raised by the call, or the active partition may deadlock. [Note: In either case, the active partition is terminated.] It is a bounded error for any deferred task activation to fail. The Environment task and all tasks whose activations fail are terminated. A task whose activation succeeds may continue to execute, or may instead be immediately terminated, thereby completing execution of the partition. [Note: The preference is to terminate the partition immediately to mitigate the hazard posed by continuing execution with a subset of the tasks being active. However this would introduce code into the runtime to detect this corner case, so it is not mandated.]

4.2 Task_Activation => Standard

This value defines the execution of the declarative part of the Environment task to be as defined by the standard mode of operation with respect to task activation and interrupt handler attachment.