Session Summary: Future of the Ada Language and Language Changes such as the Ravenscar Profile

Chair: Alan Burns
Rapporteur: Ben Brosgol

1. Status of the Standard

Jim Moore, the convener for WG9, described the formal process that is being used to update the Ada language standard. By way of background, WG9 is one of many Working Groups under SC22 (Languages and Operating Systems), which is one of several subcommittees under JTC1, which is the single Technical Committee jointly managed by both international standards organizations, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). National bodies are members at each level of the hierarchy; generally a national body is under government auspices, but ANSI (for the U.S.) is an exception. At each level of the hierarchy, national body membership and representation is determined distinctly. A country can be a P-member ("participating") or an O-member ("observing). P-members must vote on everything, and P-membership generally entails more of a commitment than O-membership. Currently WG9 has 7 P-members: Canada, France, Germany, Japan, Switzerland, UK, and US. Since new work items require commitment from at least 5 P-members, WG9 does not have much "management reserve". Alan Burns observed that other countries represented at IRTAW-11 (such as Italy, Portugal and Spain) might want to get involved.

The Rapporteur Groups under WG9 are the ARG, HRG, and ASIS. There is a potential issue at the JTC1 level since JTC1 is insisting that Rapporteur Groups properly represent national bodies. WG9 is looking into establishing formal liaison relationships with Ada Europe and ACM SIGAda, and this may help address the JTC1 issue.

A product of ISO-IEC is in one of three categories: an International Standard (which is normative), a Technical Specification (a work that either failed to achieve consensus (previously known as a "Type 1 Technical Report") or that is in progress (previously known as a "Type 2 Technical Report")), and a Technical Report (material not suitable for standardization, previously known as a "Type 3 Technical Report")).

The process of developing a new International Standard begins with a Preliminary New Work Item prepared by a WG, and submitted to the parent SC as a New Work Item Proposal. If/when approved by the SC, the WG begins the development of a Working Draft. The WG makes necessary revisions and submits to the SC as a Committee Draft. An iterative balloting process, involving the SC, results in a Final Committee Draft. This is submitted to JTC1 as a Final Draft International Standard. When approved, it becomes an International Standard. The minimum elapsed time for this entire process is one year; this was accomplished for the ACATS procedures.

In addition to the process just described, there are several other mechanisms defined by ISO. One is the "fast track" process, which has recently been initiated for C#. In this case a spec is proposed to be accepted "as is", with a vote taken at the JTC1 level. Members can vote yes/no/abstain and may include comments. In order to be approved, the spec needs to obtain 75% of the vote.

Another process, and the one currently being used for Ada, is to prepare amendments to the standard versus preparing a new standard. WG9 chose this since (1) it provides a more focused effort, and (2) it avoids the risk of opening up the entire language spec to questions. Other WGs (e.g., for SQL) have also adopted the amendments approach. SC22 has approved the project to prepare an amendment, and Randy Brukardt and Pascal Leroy have been chosen as editors. The ARG will advise the editors about the content, and WG9 will vote. This formal process is complemented by the informal way that some of the work has evolved (i.e., the Ada community's submittal of AIs to the ARG) and the combination helps in building a consensus for the eventual product.

The amendment will comprise text in the style "In place of paragraph so-and-so put new paragraph such-and-such". Thus the official Ada standard will comprise the Ada 95 standard, coupled with the Technical Corrigendum and the Amendments. There will not be a new Ada 05 standard in the sense in which Ada 95 replaced Ada 87. There are potential copyright issues that Jim is aware of and attempting to address, so that the amendments will be freely (in both senses of that word) available. Jim noted that ANSI is currently selling for $18 standards for which ISO is charging dearly, so apparently there are ways to solve the Intellectual Property issue.

Jim noted that the work on the amendments is short of staff and comprises just a few volunteers that there is little money available from DoD or elsewhere, and that commercial support is thin. This puts into perspective the likely scope of the amendments. In particular, the amendments may be able to offer, "fixes to language problems" and agreed-on extensions to the Annexes, but extensions to the core run the risk of "breakage" (unintended consequences). In the ensuing discussion, several points were made:

- Extensions to the core may be needed to fix language problems;
- It doesn't make sense approving an amendment that vendors are unlikely to implement.

Current plans are to complete the amendment work in late '03 or early '04, with final approval in '04.

2.0 Open Real-Time Issues

This discussion was comprised of several issues:

1) The proposal embodied in AI-249
2) Several issues related to the AI:
    a) Termination of tasks
    b) Task activation control
    c) Over-specification / redundancy of restrictions
    d) Boolean literals in `Simple_Barriers`
    e) Specification of the max number of tasks
    f) The `No_Task_Attributes_Package` restriction
3) Experience using Ravenscar
4) HRG report status

2.1 AI-249 – `pragma Profile`

The AI defines a new **`pragma`** `Profile`, and a specific incarnation of this pragma where the profile id is Ravenscar. The effect of this pragma is embodied in 9 existing restrictions, 10 new restrictions, a requirement to detect several bounded error conditions, `FIFO_Within_Priorities` as the default task dispatching policy, and `Ceiling_Locking` as the default locking policy. A conforming implementation cannot require additional restrictions to be implied by the `pragma` (but presumably an implementation can require that a user explicitly specify any additional restrictions via a Restrictions pragma, if the implementation requires such additional restrictions to be present).

One issue that needed to be resolved was how to specify minor variations of the Ravenscar profile; e.g. with a FIFO non-preemptive dispatching policy. There are three main alternatives:

1) "Cocktail" of pragmas:
```
pragma Profile( Ravenscar );
pragma Task_Dispatching_Policy(
          FIFO_Nonpreemptive );
pragma Locking_Policy(
          Ceiling_Locking );
```

2) Additional arguments to the pragma Profile.
```
pragma Profile( Ravenscar,
 Task_Dispatching_Policy =>
             Fifo_Nonpreemptive,
 Locking_Policy  =>
             Ceiling_Locking );
```

3) Additional profile identifier.
```
pragma Profile(
      Ravenscar_Nonpreemptive );
   -- example of identifier
```

The vote: 2-0-0 for (1); 2-0-0 for (2); 11-0-0 for (3). Thus **`pragma`** `Profile( Ravenscar )` enforces the FIFO_Within_Priorities and Ceiling_Locking policies.

The profile arguments are intended for use with other possible profiles in the future. An implementation is not allowed to add variations on **`pragma`** `Profile( Ravenscar )` with profile arguments.

2.2 Other issues related to the Ravenscar Profile

2.2.1 Task termination

The original version of the Ravenscar profile required that tasks not terminate; this restriction was felt to assist the implementer. However, the restriction cannot be enforced statically, it seems not to help the implementer, and it may be overly constraining in practice. If tasks are allowed to terminate, then a useful idiom is for a "shepherd" task to periodically check `T'Terminated` for each `task T` in its "flock" and then take appropriate action if `True` is returned.

Several alternatives are possible regarding what should be required for the Ravenscar profile:

1) Ada 95 semantics (no restrictions on whether tasks can terminate).

2) Prohibit uses of `T'Terminated` and `T'Callable` (thus a Ravenscar program won't be able to detect whether tasks have terminated, since all tasks are global and since tasks can't have entries).
3) Require that `T'Terminated` always return `False`, and `T'Callable` always return `True`

The vote: (1) by acclamation (15 in favor, none in favor of (2) or (3)).

Question: can an implementer implement termination by suspending a task forever? The answer: yes it is a conformant implementation technically (although it implies that the environment task never terminates), so users should ask the vendors how task termination is implemented if this matters.

2.2.2 Task activation control

Discussion was deferred; see 3.1 (Partition Elaboration Policy) below.

2.2.3 Over-specification of restrictions

Some restrictions are implied by others. This is not a problem.

2.2.4 Barrier conditions

Should Boolean literals be allowed

Arguments against:

- An entry with a `True` barrier is equivalent to a procedure (since requeue is prohibited).
- An entry with a `False` barrier is dead code.

Arguments in favor:

- A pre-processor may generate the Boolean literals.
- In general, unless there is a compelling reason to prohibit something we should allow it.

Consensus: allow Boolean literals. Note that Boolean variables in barriers must be components of the protected object; it is not permitted to reference global Boolean variables, even ones declared as `Atomic`. Also note: if a Boolean field is in a record that is a component of a protected object, such a field is not allowed as a barrier expression, since the AI uses the term "component" and not subcomponent".

2.2.5 What about `Max_Tasks`?

The Ravenscar profile definition is silent about the `Max_Tasks` restriction. A vendor may place an upper bound on the number of tasks. If so, then the user needs to explicitly provide a **`pragma`** `Restrictions( Max_Tasks => n )` for some value of n less than or equal to the implementation's permitted max. Note that this must be explicit; it cannot be an implicit effect of the **`pragma`** `Profile( Ravenscar )` (see 2.1 above). Since the number of tasks is dynamic (it can be the length of a run-time computed array)) exceeding `Max_Tasks` is a run-time error, presumably `Storage_Error` (for consistency with D.7(15)).

2.2.6 The `No_Task_Attributes_Package` restriction

Apparently Tucker challenged the restriction, under the assumption that it was not strong enough, but this concern is not warranted (i.e., the restriction rules out all semantic dependences on the `Task_Attributes` unit).

2.3 Experience using Ravenscar

2.3.1 Experience with Aonix's RAVEN

David Brach described his experience using RAVEN. One issue was that in the absence of a standard definition of the Ravenscar profile, vendors added implicit restrictions. He asked why enumeration type maps were prohibited in RAVEN. George Romanski explained that functions returning unconstrained arrays (such as String) used an auxiliary stack, and that there was an issue regarding the release of storage on the auxiliary stack.

David said that he found that he was using a large number of protected objects, but it's possible that an alternative approach (fewer protected objects, more fields per protected object) would also have worked.

There was a question of Ravenscar's stance toward exceptions. Alan said that since there was no consensus (in the High Integrity community) on how/whether exceptions should be used, Ravenscar is silent on the issue. Stephen Michell observed that since we require Program_Error to be raised in some circumstances, Ravenscar is not completely silent.

2.3.2 Writer / Readers Idiom in Ravenscar

Steve Michell described the issues associated with programming a typical concurrency idiom (single writer, multiple readers) using only the Ravenscar subset of the

Ada concurrency model. The main problem was the restriction that no more than one task can be queued on a protected entry. This implies the need for multiple protected objects (one per reader), a style that was somewhat heavy (writer needs to alert all readers when data is present). Tullio Vardanega pointed out that it was possible to separate out the scheduling logic from the writer logic. Steve said that we need "Ravenscar patterns" so that users don't have to reinvent common idioms.

2.4 HRG Report on Ravenscar

Alan gave a status update. The current draft is close to 60 pages including examples, rationale, motivation, definition (including the AI), etc. The aim is to be of general utility to the Ada community. It may be published first by the University of York, before it is officially submitted to the HRG. It is expected to be complete within around six months.

3. Other AIs

Several general Ada AIs (i.e., not Ravenscar specific) were discussed.

3.1 AI-265: Partition Elaboration Policy

The **pragma** Partition_Elaboration_Policy ("PEP" for short) has been proposed to address a problem with Ada 95 elaboration order. (The problem is complexity of analyzing programs, and complexity of implementing the required run-time support, if elaboration of library-level tasks occurs piecemeal during package elaboration, and if interrupts can occur during package elaboration.) Under pragma PEP(Sequential), library task activation and also interrupt delivery are deferred until the "begin" of the environment task.

There was a general feeling that this capability was needed (although it wasn't clear what would happen if an interrupt did occur during the elaboration stage), but there was a vigorous discussion of whether PEP(Sequential) was needed just for Ravenscar, or for the full language.

Arguments supporting "just for Ravenscar":

- This is the community for which it is needed.
- There are complicated semantics if try to extend to full Ada (e.g. what happens with tasks that are dynamically allocated during package elaboration) and it may be a large effort to get all the details defined.

Arguments supporting "apply to full Ada"

- In all other aspects, Ravenscar is a subset; here the different elaboration semantics are an extension and there is concern that the Ravenscar profile proposal may be rejected if it is intrinsically associated with a changed piece of core language semantics.
- May be worthwhile in other environments besides Ravenscar / High Integrity.
- Previous IRTAW considered the problem and recommended this approach

Consensus (13.5 in favor) was that PEP(Sequential) would be a separate proposal from the Ravenscar Profile, but that it would apply only in the presence of pragma Profile(Ravenscar). Although this is equivalent to defining a profile argument for pragma Profile(Ravenscar), the latter has the disadvantage of highlighting the association of Ravenscar with a language extension. Note that separating the proposals means that an implementation can comply with the Ravenscar requirements without implementing PEP(Sequential).

[During the discussion the following statement in AI-265 was thought to be in error, since the environment task obviously needs to be suspended while the library level tasks are activated: "... it is a bounded error for the Environment task to execute a potentially-blocking operation other than a delay statement during its declarative part." This was felt to be in error since obviously the Environment task needs to block while the library level tasks are activated. But this activation occurs after the "begin" of the environment task, not during its declarative part. Thus the AI need not be corrected in this area.]

3.2 AI-266: Task termination procedure

This AI proposes a mechanism ("task group") through which the application programmer can provide "termination hooks" for procedures that are called under a variety of circumstances related to task termination (unhandled exception, abort, normal termination, never activated). The task group concept is based on the "thread group" notion in Java. The AI stems from a discussion at the Exceptions workshop at Ada Europe 2001.

Several issues were discussed:

1) Is "silent task death" a problem that needs to be solved?
2) Is anything special needed (i.e., can controlled types be used)?
3) Is the Java thread group an appropriate basis for the design?

Regarding (1), there was general support for the need for such functionality (e.g. for fault tolerance or "health monitoring"); the vote was 13-0. Note: this vote applied only to the unhandled exception hook, not to the additional items (abort etc.), although subsequent discussion showed support for these also.

Regarding (2), controlled types alone are not sufficient. Finalization for a controlled object occurs when the object is destroyed, but (e.g. in Ravenscar) such destruction will only take place when the object is unchecked-deallocated, and unchecked deallocation might be disallowed by the implementation or by application requirements. Moreover the "hook" needs to be invoked not when the object is finalized, but at the earlier point when the task terminates.

Regarding (3), there was some concern that the design was based on a Java feature which seems to have fallen into disrepute in Java. Bloch's book "Effective Java" states: "... thread groups are largely obsolete.... [they] don't provide much in the way of useful functionality, and much of the functionality they do provide is flawed. Thread groups are best viewed as an unsuccessful experiment, and you may simply ignore their existence.... If you are designing a class that deals with logical groups of threads, just store the Thread references comprising each logical group in an array or collection." Bloch goes on to observe: "There is a minor exception to the advice that you should simply ignore thread groups. One small piece of functionality is available only in the ThreadGroup API. The ThreadGroup.uncaughtException method is automatically invoked when a thread in the group throws an uncaught exception." The AI would need a much stronger rationale for the introduction of a ThreadGroup-based mechanism, given this negative experience from Java.

The vote on the specific proposal in the AI was 0-16; it was felt that the mechanism was complex and beyond the needed functionality, and that it was stylistically clumsy to use if all you wanted to do was to provide an unhandled exception procedure for a specific task. Several alternative approaches were conjectured: special task attribute procedures, and a proposal based on task-ids.

An example of a possible approach using attributes:

```
task T;  -- also OK for task type
procedure
   My_Unhandled_Exception_Procedure(
           Id : Task_Id;
            E : Exception_Occurrence);
for T'Unhandled_Exception_Hook
use My_Unhandled_Exception_Procedure;
```

A drawback of using special attributes is that it affects the compiler; however, the original AI might also affect the compiler in terms of the need for calls to be inserted at special places. Another drawback of the attribute approach is that it doesn't scale up to applications where tasks are allocated dynamically (at least not if the program needs to associate a different termination hook with different dynamically allocated tasks of the same task type).

A pragma-based approach has the same set of issues as the attribute approach.

The AI uses OOP and provides a flexible and extensible framework, whereas the attribute and task id approaches are somewhat more special purpose. However, OOP may be a disadvantage in some communities, particularly the High Integrity domain, so the use of OOP is a double-edged sword.

There was agreement that whatever mechanism was adopted should be applicable both to the full language and to restricted environments such as Ravenscar.

Andy Wellings presented an alternative proposal using task ids rather than OOP. This was part of the session on Wednesday afternoon but is included here since it was a continuation of the discussion on task termination. It is intended to be applicable both to full Ada and to Ravenscar:

```
package Ada.Task_Termination is
  type Termination_Handler is access
    procedure(
            Id: Task_Id,
        Except: Exception_Occurrence);

    procedure
      Unhandled_Exception_Set_Handler(
        Id  : Task_Id := Current_Task;
        Handler: Termination_Handler);

   --Similarly, for other termination
   --events of interest, e.g. normal
   --termination,  abortion
end Ada.Task_Termination;
```

An example of usage:

```
package body Termination_Logging is
  procedure Log_Non_Normal_Termination
     (Id: Task_Id;
      Except: Exception_Occurrence)is
  begin
    --write out error message to
    --operator terminal.
    --log event to file
  end Log_Non_Normal_Termination;
end Termination_Logging;

with ...; use ...;
package body Ravenscar_Example is
  task MyTask;
  task body MyTask is
  begin
   Task_Termination.
      Unhandled_Exception_Set_Handler
            (Handler =>
   Log_Non_Normal_Termination'Access);
       loop
           ...;
       end loop;
    end MyTask;
  end Ravenscar_Example;
```

After the workshop, Andy circulated a revised version of this proposal.

The discussion focused on several issues:

- Is it more applicable to Ravenscar than to the full language?

  One can always use exception handlers to catch exceptions that would otherwise cause the task to die silently, so `Unhandled_Exception_Set_Handler` is not really necessary (it would be applicable in environments where exception handling is restricted or prohibited).

- Use interrupt handling model?

  The **package** `Ada.Interrupts` (and perhaps the associated attachment-related pragmas) provide a possible model for setting the task termination hooks. However, the uses are likely to be different for interrupt handler attachment/detachment than for task termination hook set/reset. Although it is fairly common to need to dynamically replace interrupt handlers, the task termination hooks are almost always set once and never changed thereafter.

- Race conditions

  Setting handlers dynamically risks race conditions, especially if the handler is set in other than the affected task. Thus usage of the "hook setting" procedures will require care. This seems also to be an issue with the OOP approach proposed in the AI. An alternative design with attribute procedures or pragmas seems to avoid the race condition problem, but (as noted above) it does not scale up to full language constructs; e.g., how to establish termination hooks for a dynamically allocated task.

- What should happen if a termination hook is set after the task has terminated?

  Several possibilities:
  1) Invoke the termination procedure immediately.
  2) No op.
  3) Raise an exception (probably `Tasking_Error`).

  Alternative (1) has several problems. First, it doesn't answer the question of what happens if the hook being set is not for the circumstance that caused the task to terminate (e.g., the task terminated normally, and you try to set the unhandled exception hook). Second, it requires the implementation to store additional information (cause of termination) for a terminated task. Third, having the hook setting task invoke the termination procedure is different from the situation in all other cases, where the hook is called by the task that is about to terminate.

  Alternative (2) is not attractive; setting a termination hook for a task that has terminated is an error that should be detected. Thus alternative (3) seems the best.

- Are there restrictions (in terms of language features) on what a hook procedure can do?

  It seems most appropriate to use the same restrictions as for `Finalize`. Additional restrictions were discussed (e.g. no potentially blocking operations) but this would rule out I/O calls and would complicate the style for common idioms such as logging to a file.

  The consensus of the group was to encourage Andy to flesh out his proposal as an alternative

to the OOP style in the AI, and to submit it to the ARG.

3.3 AI-250: Extensible Protected Types

Rodrigo García García gave a brief report on EPFL's experience attempting to implement this feature in GNAT. There were some problems (not surprising), and the impact of the new feature is not small, but it looks like it can be implemented in GNAT without a major restructuring of the compiler. It is unclear, however, if there is funding available for further investigation/implementation.

Steve summarized the status of this AI. Tucker disagrees with the AI's rules on overriding entries and feels that the subclass should be able to completely replace (rather than simply strengthening) the barrier conditions. The ARG did not like the different treatment of protected operations regarding dispatching: procedures and functions result in dispatching, whereas entries do not. The ARG also was concerned about some visibility issues, in particular with the ability (granted in the AI) to see the private part of the parent.

However, even if all the technical issues are sorted out, there is still the underlying question of whether to move this AI forward. On the positive side, the proposed facility does fill a gap in the language and offers a better combination of OOP and concurrency than is found in other languages (e.g., Java). However, there is no obvious user need here, the lack of funding for Randy Brukardt complicates the job of finding an editor, and there is no reference implementation. Moreover, if there is a need from some segment of the user community, that segment is not likely to be from the real-time domain, so IRTAW would not be a likely forum to drum up enthusiasm. The general sentiment was that there was no need to drop the AI, but neither was there a need to move it forward. Given the priorities, the schedule goals, and the financial constraints, this AI is highly unlikely to make it into the amendments.

3.4 AI-264: Exceptions as Types

This AI proposes a Java-like mechanism for providing two pieces of functionality missing from Ada 95: the ability to pass arbitrary data objects with an exception when an exception is raised, and the ability to define hierarchies of exceptions. It defines an exception type as an untagged composite limited type, and uses type derivation (rather than Ada 95 inheritance) to achieve hierarchies. The AI is based on the Exceptions workshop at Ada Europe 2001.

The ensuing discussion resulted in a number of points (some of which are in conflict):

- Additional capabilities in the spirit of the AI would be useful.
- The AI does not go far enough (e.g. it lacks requiring something in the subprogram signature identifying the exceptions that can be propagated).
- Additional capabilities in the exceptions area are not needed, especially for the real-time community.
- Indeed, the whole topic of exceptions is dealt with differently in different segments of the real-time community, ranging from "use the whole model" to "use only for global shutdown" to "don't use at all."
- For the safety-critical community, exceptions have several undesirable properties; e.g. searching for a handler introduces a time delay and complicates testing.
- Some technical issues need to be addressed by the AI; e.g., the meaning of reraise, whether a program can construct an exception object and later raise it.
- The rules for finding a handler have the look of an implicit conversion across derived types; such conversions would be illegal in other contexts.
- The proposal has large impact on the implementation.
- The syntax will be confusing, especially to students and new users, since it is similar to but different from tagged types.

The resulting consensus position (9-0) was:

- There is sympathy for the need for additional exception functionality.
- There is concern over the specific solution / features proposed in the AI.
- The topic is not of high priority for the real-time community.