Minutes of the 40th ARG Meeting

26-28 February 2010

Burlington, Massachusetts, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns (except Sunday), Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen (via Skype, mornings and part of Saturday afternoon), Ed Schonberg, Tucker Taft, Bill Thomas (except Friday morning and Sunday afternoon).

Observers: Greg Gicca, Matt Heaney (Saturday only).

Meeting Summary

The meeting convened on 26 February 2010 at 9:15 hours and adjourned at 15:20 hours on 28 February 2010. The meeting was held in the conference room of Sofcheck, Inc.. As expected, the meeting covered about two thirds of the entire agenda.

AI Summary

The following AIs were approved :

AI05-0150-1/03 Use all type clause (6-1-3)
AI05-0196-1/02 Null exclusion checks for 'out' parameters (10-0-0)

The following AIs were approved with editorial changes:

AI05-0031-1/02 Add a From parameter to Find_Token (7-0-2)
AI05-0049-1/02 Extend file name processing in Ada.Directories (9-0-1)
AI05-0113-1/04 Conflicting external tags and other tag issues (6-0-3)
AI05-0124-1/01 Where is the elaboration check suppressed? (10-0-0)
AI05-0144-2/03 Detecting dangerous order dependencies (5-0-4)
AI05-0145-2/05 Pre- and Postconditions (11-0-0)
AI05-0159-1/04 Queue containers (11-0-0)
AI05-0162-1/02 Allow incomplete types to be completed by partial views (10-0-0)
AI05-0166-1/03 Yield for non-preemptive dispatching (9-0-2)
AI05-0168-1/03 Extended suspension objects (11-0-0)
AI05-0169-1/02 Group budgets for multiprocessors (9-0-2)
AI05-0174-1/02 Implement task barriers in Ada (9-0-2)
AI05-0176-1/04 Quantified expressions (6-1-3)
AI05-0179-1/03 Labels at end of a sequence_of_statements (10-0-0)
AI05-0203-1/01 A return_subtype_indication cannot denote an abstract type (10-0-0)
AI05-0205-1/01 An extended return statement declares a name usable inside the statement (10-0-0)
AI05-0207-1/01 Access constant is considered for mode conformance (9-0-1)
AI05-0208-1/02 Characteristics of incomplete views (8-0-2)

The intention of the following AIs was approved but they require a rewrite:

AI05-0139-2/04 Syntactic sugar for accessors, containers, and iterators (10-0-1)
AI05-0142-4/05 Explicitly aliased parameters and accessors for Ada.Containers (6-1-4)
AI05-0147-1/09 Conditional expressions (9-0-0)
AI05-0153-1/04 Subtype predicates (6-1-3)
AI05-0167-1/02 Managing affinities for programs executing on multiprocessor platforms (9-0-2)
AI05-0170-1/02 Monitoring the time spent in Interrupt Handlers (8-0-3)
AI05-0171-1/03 Ravenscar Profile for Multiprocessor platforms (11-0-0)

The following AIs were discussed and assigned to an editor:

AI05-0111-1/08 Specifying a pool on an allocator
AI05-0117-1/00 Memory barriers and Volatile objects
AI05-0127-1/02 Adding Locale Capabilities
AI05-0135-2/02 "Integrated" nested packages
AI05-0151-1/06 Allow incomplete types as parameter and result types
AI05-0158-1/04 Generalizing membership tests
AI05-0186-1/05 Global-in and global-out annotations

The following AIs were discussed and voted No Action:

AI05-0074-1/02 Limited view of generic instantiations (9-0-0)
AI05-0074-2/01 Allowing an explicit "end private;" in a package spec (9-0-0)
AI05-0074-3/00 Deferred instance freezing (9-0-0)
AI05-0074-4/02 Private Instantiations (9-0-0)
AI05-0074-5/01 Private Instantiations - Take 2 (9-0-0)
AI05-0135-1/05 "Integrated" nested packages (9-0-0)
AI05-0139-1/02 User-defined iterators (9-0-0)
AI05-0141-1/01 User-dereferencing in Storage Pools (9-0-0)
AI05-0142-1/01 Variable function results (9-0-0)
AI05-0142-2/01 Limited access types (9-0-0)
AI05-0142-3/02 Accessors for Ada.Containers (9-0-0)
AI05-0144-1/03 Detecting dangerous order dependencies (9-0-0)
AI05-0145-1/02 Pre- and Postconditions (9-0-0)
AI05-0153-2/05 Set constraints and extended discriminant constraints (6-0-5)
AI05-0172-1/01 Extension to Ravenscar Profiles (10-0-1)
AI05-0175-1/02 Cyclic fixed point types (10-0-0)
AI05-0187-1/01 Shorthand for assignment with expressions naming target (8-0-2)
AI05-0204-1/01 String packages should be Remote_Types (8-0-2)

The following SIs were approved with editorial changes:

SI99-0050-1/01 Missing primitive subprograms in Is_Dispatching_Operation (8-0-1)
SI99-0051-1/01 Default actuals for formal procedures with null defaults (6-0-3)
SI99-0052-1/01 Undefined behavior of Corresponding_Body (7-0-2)
SI99-0056-1/01 Information about components defined by an access_definition (7-0-2)
SI99-0059-1/04 Handling enumerations with very many literals (7-0-2)
SI99-0061-1/01 Gap in Asis.Declarations.Corresponding_Declaration (7-0-2)

The following SIs were discussed and parts were approved and the intent of the remainder was approved and assigned to editors for a rewrite:

SI99-0054-1/05 Open issues in the ASIS semantic subsystem (7-0-2)
SI99-0058-1/02 More open issues in the ASIS syntactic subsystem (8-0-3)

The intention of the following SIs was approved but they require a rewrite:

SI99-0049-1/01 Asis.Ids should be obsolescent (7-0-4)
SI99-0060-1/01 Add an example of a ASIS semantic subsystem (7-0-2)

The following SI was discussed and voted No Action:

SI99-0063-1/01 Normalized pragma argument lists (7-0-2)

Detailed Minutes

Previous Meeting Minutes

John and Jean-Pierre sent in editorial fixes, they have no yet been applied. None are very significant.

Approve minutes by acclamation.

Date and Venue of the Next Meeting

The next meeting is associated with Ada Europe, Valencia Spain, June-17-20.

Do we need an extra meeting before then? We need to have ASIS finished before Valencia, so we might need a meeting in April. We probably ought to have separate ASIS and Amendment meetings. The phone meetings have worked reasonably well, so we will try to have phone meetings for both groups.

ASIS Review

We need to set the dates for the upcoming ASIS review. The editors will distribute the document and assignments no later than March 22. The reviews will be due at end of April. The ASIS phone meeting will need to be about a week before that. On Sunday, we set the date of the ASIS phone meeting to March 12th, time to be determined after checking with Sergey Rybin and Gary Barnes.

RM Review

We also need to set the dates for the upcoming AARM review. We don't want that to overlap too much with the ASIS and AI editorial reviews. So we'll try to distribute the document and assignments in late April. The reviews will due in late May.

Amendment Big Picture

We need to look at the big picture for the Amendment. There is some concern that we are getting too many changes without looking at the overall impact.

Randy started to create a spreadsheet of the AIs, but he ran out of time to provide anything useful. We'll delay this discussion until the end of this meeting [but we didn't discuss it then – Editor].

Thanks

Thanks to Tucker Taft for providing the conference room and refreshments. Thanks to his wife for hosting the party on Saturday night.

Old Action Items

Pascal didn't do his task. Erhard didn't finish finding his examples of aliasing; his sample code didn't compile. Ed offered help with that. Tucker flooded his basement and didn't finish much; he only finished AI05-0144-2 (Randy actually did this one), AI05-0139-2, AI05-0111-1, and most of SI99-0054-1 were finished.

New Action Items

The combined unfinished old action items and new action items from the meeting are shown below.

All ARG members:

- Report on possible uses of AI05-0111-1.

Steve Baird:

- AI05-0135-2 (no actual work required at this time)
- AI05-0144-2 (improve wording for slices).
- AI05-0147-1
- Help Ed Schonberg with AI05-0158-1.
- Create a new proposal for formal incomplete types (with help from Ed Schonberg), see discussion of AI05-0135-2.
- Participate in the amendment subcommittee.

Randy Brukardt:

- Craft an AARM note/link for the syntax changes of AI05-0147-1 (see discussion)
- AI05-0142-4 (with help from Tucker Taft)
- AI05-0186-1
- SI99-0049-1
- Participate in the ASIS and amendment subcommittees.

Editorial changes only:

- AI05-0031-1
- AI05-0049-1
- AI05-0113-1
- AI05-0124-1
- AI05-0145-2
- AI05-0159-1
- AI05-0162-1
- AI05-0166-1
- AI05-0168-1
- AI05-0169-1
- AI05-0174-1
- AI05-0176-1
- AI05-0179-1
- AI05-0203-1
- AI05-0205-1
- AI05-0207-1
- AI05-0208-1
- SI99-0050-1
- SI99-0051-1
- SI99-0052-1
- SI99-0054-1 (completed parts)
- SI99-0056-1
- SI99-0058-1 (completed parts)
- SI99-0059-1
- SI99-0061-1

Alan Burns:

- AI05-0117-1
- AI05-0167-1
- AI05-0170-1
- AI05-0171-1

Bob Duff

- AI05-0153-1
- Participate in the amendment subcommittee.

Greg Gicca:

- Participate in the ASIS subcommittee.

Pascal Leroy:

- Create AI to combine wording in AI05-0006-1 with other parts of the language (from his editorial review).

Brad Moore:

- AI05-0127-1

Jean-Pierre Rosen:

- Participate in the ASIS subcommittee.

Ed Schonberg:

- Study whether the proposed solution for AI05-0049-1 would make sense for VMS.
- AI05-0158-1 (with help from Steve Baird)
- Help Steve Baird with the proposal for formal incomplete types (see discussion of AI05-0135-2).
- Participate in the ASIS and amendment subcommittees.

Tucker Taft:

- AI05-0111-1
- AI05-0122-1
- AI05-0139-2
- AI05-0146-1
- AI05-0151-1
- AI05-0155-1
- AI05-0183-1
- AI05-0189-1
- Help Randy Brukardt with the accessibility rules for AI05-0142-4.
- Handle various changes now in SI99-0058-1, see minutes of St. Petersburg meeting.
- Handle various updates to parts of SI99-0054-1 (see minutes).
- Participate in the ASIS and amendment subcommittee.

Bill Thomas:

- Handle various changes now in SI99-0058-1, see minutes of St. Petersburg meeting.
- SI99-0060-1
- Set up an ASIS subcommittee conference call to handle remaining ASIS open issues for March 12 or so (see discussion of SI99-0054-1); participate in that call.

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs) and Ada 2005 AIs. The AIs and SIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Final Consolidated AARM. Paragraph numbers in earlier drafts may vary.

Detailed Review of ASIS Issues

SI99-0049-1/01 Asis.Ids should be obsolescent

Jean-Pierre says that he would never trust such functionality to work. Bill says that they did use it in ASIS 1.1.

The GNAT implementation makes no attempt to make this work usefully. There is no guarantee that even the exact same source code would allow Ids to work. The IBM implementation does make this work, of course.

Randy notes that if we are not going to make this obsolescent, then we need to say something about what is expected to work so that ASIS users have guidance about what they can depend on.

Tucker suggests saying that it is implementation-defined what changes to the context would prevent Ids from matching. That would require that implementations document what works (which might be nothing beyond the exact same source). The exact same source is expected to work for all implementations.

Randy will rewrite this AI to do that.

Approve intent of SI: 7-0-4.

SI99-0050-1/01 Missing primitive subprograms in Is_Dispatching_Operation

Add "(Yes.)" to question, and only one question mark.

Approve SI with changes: 8-0-1.

SI99-0051-1/01 Default actuals for formal procedures with null defaults

Fix wording (2nd paragraph of 18.xx): "Returns True {if}[is] Statement"

Fix title of 18.xx to match the name of the function. (The name of the function is correct.)

Drop the "in the expanded instance", "in the instance" from the second wording.

Fix question: "…defining the semantic{s} [of the meaning] of the formal…"

Approve SI with changes: 6-0-3.

SI99-0052-1/01 Undefined behavior of Corresponding_Body

We should have an AASIS note saying that it is unspecified what is returned for the body of an instance if the body of the corresponding generic is provided by pragma Import.

Gary asks that the subject mention generic instances. "Corresponding_Body for instances of imported generic units".

Approve SI with changes: 7-0-2.

SI99-0054-1/05 Open issues in the ASIS semantic subsystem

We discuss the Gary Barnes comment to "delete the semantic subsystem". He said that he didn't think it was practical to implement it in the IBM ASIS.

Sergey has previously said he doesn't want to implement it because he can already get the information through a private interface. But that doesn't address portable access for user programs – applications shouldn't be tied to Sergey's private interface.

We think users will demand this functionality. Erhard notes that he knows of 5 projects that tried to use ASIS and the use of ASIS was abandoned (or the whole project was abandoned) because of insufficient semantic information.

Tucker notes that there is no requirement in the semantic subsystem for persistent state; the results can be calculated on the fly if necessary. So it isn't necessarily implying a new form of ASIS data.

We believe that this is important new functionality and we intend to keep it in the standard.

Steve says that we would like to have specific concerns about implementability of particular interfaces explained. It's easy to believe that there are particular problem areas, but we need detailed comments.

Moving on to the detailed changes. The only areas of change from last time is that Randy put in the specific wording we agreed to last time, and Tucker has proposed additional wording to answer most of the remaining questions. We decide to just look at Tucker's changes.

Wording change for 23.14.1: modify the change " represents a view of a {named} statement" to make it clearer that these are named entities (only). Probably add an AASIS note for additional clarification.

The wording for 23.2.2 and 23.16.1 seems OK. Why would you use this? One reason would be to display an answer, such as the identifier of an entity.

23.2.1: The text is OK, except that "Is_Object_or_View" should be "Is_Object_or_Value" in the original text. Add the suggested example.

23.2.2: Jean-Pierre would like a predicate to tell if two views denote the same thing. "=" means the elements associated with the views are the same. (That is handled in SI99-0062-1, which we didn't discuss at this meeting.) Tucker asks exactly what "same" means here. Jean-Pierre isn't able to say precisely; he gets an action item to answer this and propose functions.

Second item:

Jean-Pierre wonders why you could never get a defining name. Can you get the defining name of a statement (a label)? For a label, you can only see it through a usage name (we have no way to come from a statement). A usage name is never a defining name.

Tucker's new wording seems to cover this topic.

23.2.5: For Has_Declaration, Jean-Pierre would like to know when it could return False, so he would like an example in the text.

"For example, views representing X.all, 2+3, A(I) don't have declarations."

We turn to the question about Defined_View. Tucker means "object declaration" in this wording to be expansive; it includes formal parameters and components. But access results of functions also need to be covered somehow. It probably would return the entire function.

Tucker believes Has_Declaration is still incorrect; he would rather revert to a version closer to the original version. He gets an action item to do that. Randy notes that the problem of finding the location of an anonymous type or subtype (the former is far more important) still exists and needs to be solved.

Steve asks if there is an issue with extended_return_statement. It is a declaration so it is OK.

23.2.6: Typo: "The Public_Child_Part and the Private_Child_Part [include] might not include child units"

The Callable_Formal_Part is a visible part; it is the profile minus the return type.

This looks OK.

23.5.1: Tucker says that this wording isn't quite right; he'll take an action item to fix that. (An object_view could be something else other than an object declaration.)

23.5.2: Randy worries that the proposed wording doesn't answer the overall renames question. Tucker thinks that this is a weird case, because it is structural. He thinks that usually it isn't an interesting question; for most properties renaming doesn't make any difference. Randy will take an action item to review the rest of 23 (yuck) to find other cases where renaming is interesting. Otherwise OK.

23.5.3: Changes are OK. Add an Is_Explicit_Dereference query to be a counterpart of Is_Implicit_Dereference.

23.8.1: Steve wonders what happens for a class-wide limited type? Should it always return True as there *might* be an extension that contains a task? Tucker thinks that it should return the answer of the specific type; that would provide more information. Tucker will take an action item to fix that.

For the third item, Tucker starts reading the wording he didn't previously provide. We'd rather be able to read it ourselves; he will provide it with his action items.

Should Nondiscriminant_Region_Parts return a list of parts or one giant part? It's easier to ignore information than to add more information, so it should be a list of parts. Steve asks are empty (extension) parts included? (Yes, keep the number of returned items the same as the number of extensions.) Bob asks if private components are included? (Yes, seems necessary.)

23.12.4: Current_Instance. Tucker says that this allows you to use the queries for subprograms and packages on the generic unit. It allows us to not duplicate all of the functionality of subprograms and packages for generics. We don't have a similar need for task bodies or record declarations, thus we don't need equivalents there.

Tucker should add a user note to explain this. (Editor's musings: Aren't we being too pedantic if we won't consider a generic subprogram a subprogram, and a generic package a package? Just because the language standard insists on that rather un-intuitive definition doesn't mean that we have to copy it here.)

The last item is still assigned to Tucker Taft.

Approve completed parts of SI: 7-0-4.

We will try to set up an ASIS subcommittee conference call; Bill will take the lead.

SI99-0056-1/01 Information about components defined by an access_definition

"Returns an element that has {one of} the…"

Gary notes "(no)." should be "(no.)".

Approve SI with changes: 7-0-2.

SI99-0058-1/02 More open issues in the ASIS syntactic subsystem

2.3: Greg has send Randy updated graphics for the Standard. That happened after the current draft, but they should appear in the next one.

3.10.1: Do this.

Section 9: Bill asked Sergey about this, and he also reported a 1-to-1 mapping of containers to contexts.

Even if this was implemented to represent projects or the like, there is very little useful that could be done portably with the interface. The names of containers are implementation-defined, and the only defined operations are to get the contained compilation units. An application would need to understand the environment structure of a particular implementation to learn anything interesting, and that is by definition implementation defined.

So this type and the related queries don't appear to be useful; make them obsolescent.

13.4: There is a query for getting pragmas that apply to a unit: Compilation_Pragmas. We're not sure why the original Gary Barnes comment did not suggest using that standard interface rather than an implementation-defined one.

In any case, Tucker thinks this is useful. He suggests adding a note to clarify that it is talking about the contents of the "initial" context (perhaps the main subprogram); units compiled into other locations might have different pragmas (which can be retrieved using Compilation_Pragmas).

Tucker will e-mail some wording.

17.1: Use the wording suggested in the SI.

Jean-Pierre would like the second bullet to say something about the correct type. That is, "an implementation-defined declaration of a subtype of the appropriate type". Drop the second note.

17.7: Ed will do this. Tucker notes that action items for ASIS ought to be distributed as soon as possible (before the minutes) with a March 15 deadline.

17.21: OK.

17.22: Tucker says that he needed help. There are no actual parameters for formals in the parameter list of a formal package parameter. (Those come from the instance of the actual package.)

We decide the question is about normalized actual parameter lists of the *formal package parameter* of a generic.

The normalized list would have all named parameters, and **others** => <> gets expanded into one parameter per <>.

This item is still assigned to Tucker.

19.1: Fix as suggested.

20.13: Jean-Pierre claims that this is just an equals function, comparing against Nil_Span. But that is very weird, values of the record can be created with an aggregate.

The wording should be changed to "empty span", where an "empty span" is defined as "the location associated with upper bound precedes the location associated with the lower bound in the text".

[Editor's note: Wording based on "single text position" is better as that is already a defined term.]

22, 22.1, 22.19: Bill will take the action item.

D.4: Still assigned to Tucker.

F.3.2: Typo: obsolescent is misspelled.

How commonly used is this interface? Jean-Pierre says that he prints Diagnosis in his bug box.

Tucker thinks that we should keep it because it is used commonly.

This isn't task-safe. But neither is the rest of ASIS, so that isn't particularly relevant.

So we decide to put it back and revert it to the original text. We should keep the improved wording in 1.1.4 and 4, adjusting it to include Diagnosis.

Approve the parts of the SI that are finished (except 13.4 note, 17.7, 17.22, 22 [all], D): 8-0-3.

SI99-0059-1/04 Handling enumerations with very many literals

Remove the change notes from the discussion section.

"...not have one of these expected kinds." Use the singular version of this wording. "...not have this expected kind.".

"Returns" should be "Corresponds to" for Enumeration_First_Pos.

Asis_Positive ought to be Asis_Natural. A type with a single enumeration literal has Last_Pos = 0.

Approve SI with changes: 7-0-2.

SI99-0060-1/01 Add an example of a ASIS semantic subsystem

Typo in the subject: "Add an example illustrating use of the ASIS semantic subsystem".

Bill has some improvements to the example; he'll provide them separately.

The call should be "Asis.Callable_Views.Declaration", or use prefix notation "Call_View.Declaration", as these declarations are inherited. The latter form is preferred, for all of these calls.

Bill objects that this would make it harder to find the actual declaration. The other examples have cross-references for each call to the actual declaration; this one ought to as well. Add "-- 23.2.11" in the example above.

Make the clause B.3 instead of B.2.1.

Approve intent: 7-0-2.

SI99-0061-1/01 Gap in Asis.Declarations.Corresponding_Declaration

…this [contradicts]{conflicts} with the definition of the returned kinds list for Corresponding_Declaration.

Approve SI with changes: 7-0-2.

SI99-0063-1/01 Normalized pragma argument lists

We would have to define what the default parameters are for all of the pragmas., as the Ada Standard has no such idea. That's too much work, so drop that idea from the SI.

We discuss what a normalized parameter would look like. If there is a name for a parameter, it would appear. The parameters would not be reordered (Ada doesn't allow changing the order of pragma parameters).

It's not obvious that any of this is worth the effort. There aren’t names in many cases, so it isn't possible to depend on the names being present. And there is no reordering of parameters to worry about, and we previously decided defining default values is too hard.

No Action: 7-0-2.

Detailed Review of Ada 2005 AIs

AI05-0031-1/02 Add a From parameter to Find_Token

A null string will always raise Index_Error. That's inconsistent and unintended. Change the first sentence of the wording:

If Source is not the null string and From is not in Source'Range, then Index_Error is raised.

Approve AI with changes: 7-0-2.

AI05-0049-1/02 Extend file name processing in Ada.Directories

Change the name of function Is_Root_Directory to Is_Root_Directory_Name to be consistent with the others.

Drop Full_Name, Base_Name and Extension from the child package, because we can use the ones in Ada.Directories (which are identical).

Is_Simple_Name should be a renames of the one in Ada.Directories. Same with Simple_Name. Umm, there is no Is_Simple_Name in Ada.Directories, so that's impossible. Probably do that for Containing_Directory, though.

Improve the wording for Name_Case_Equivalence: "…of the file name {used} when a file is created; and Case_Insensitive otherwise. Returns Unknown if [it]{the file name equivalence} is not known [which answer is correct]."

Tucker suggests: "Containing_Directory returns the [Containing_Directory]{containing directory}; it raises Use_Error if Name does not [identify]{have} a containing directory (this includes if any of Is_Simple_Name, Is_Root_Directory, Is_Parent_Directory_Name, or Is_Current_Directory_Name are True)."

[But we don't need the above text, as we changed Containing_Directory to a rename. - Editor.]

Approve AI with changes: 9-0-1.

AI05-0074-1/02 Limited view of generic instantiations
AI05-0074-2/01 Allowing an explicit "end private;" in a package spec
AI05-0074-3/00 Deferred instance freezing
AI05-0074-4/02 Private Instantiations
AI05-0074-5/01 Private Instantiations - Take 2

Randy had separated out a list of older AIs (mostly discarded alternatives) that he expected to eventually vote no action.

We look at the list, and don't see anything we want to keep around. In addition to the ones above, the list includes AI05-0135-1, AI05-0139-1, AI05-0141-1, AI05-142-1, AI05-0142-2, AI05-0142-3, AI05-0144-1, and AI-0145-1 (see individually).

No action on all of these: 9-0-0.

AI05-0111-1/08 Specifying a pool on an allocator

Tucker notes that Steve has found two bugs. New_Strong_Reference and New_Weak_Reference should take parameters of type Acc.

Randy and Gary wonder why we are abbreviating Acc and Desig. Bob notes that **access** is reserved. Unchecked_Deallocation uses Object and Name for this, this seems like it should be similar.

Tucker begins to explain the model.

The storage pool for an access type is logically partitioned into subpools. When the last reference to a subpool goes away, the subpool's objects are finalized and deallocated.

Objects have to "know" somehow which subpool they are allocated from. Deallocation does not take a subpool parameter (it can get it from the allocated object). This implies some overhead and is a requirement on the implementer of a pool with subpools.

Steve suggests that the Postcondition for Allow_Reference is that Is_Reachable (From, To) is True.

All allocations out of a Subpool_Manager pool allocate from some subpool; there isn't any allocations from a "default" pool without using the subpool.

There is an extra "end System.Subpools;" right before **private**.

Randy wonders if the Create_Subpool operation being required is a good idea, since he says that the manager might require additional parameters – say a location for a persistent subpool. Tucker says that he would like an operation that works for all subpools; a particular implementation might add more of it's own. You can always add more operations, it's unusual to encourage it. That implies that the provided one is insufficient; if true, then the provided operation is just noise.

Tucker notes that subpools shouldn't be allowed to outlive the underlying pool.

Subpools.Implementation is the interface between the compiler and the subpool manager for the finalization stuff. Some people find this confusing.

For a subpool, the Ada Implementation does reference counts, finalization, task dependence, reachability checks, initialization of allocated objects; and the subpool implementation does storage allocation and reclamation, and keeps track of which subpool a given access value points at.

The Ada Implementation provides the "collection" view of allocated objects; the subpool implementation provides the "pool" view of the allocated objects.

Bob goes back to why the subpool handle isn't passed to Unchecked_Deallocation. Tucker says that only the implementation can figure that out correctly. An access object can hold objects allocated from several different subpools. [It would be possible for the implementation to keep this information, by using fat pointers for instance. It would be helpful to describe why this shouldn't be done. - Editor.]

The storage pool has to have the same accessibility as the access type. That seems like a weird restriction; Tucker thinks it is necessary to prevent the subpool from outliving the access type. Why not just say that? Probably because the subpool object doesn't actually exist visibly. (Randy wonders why that is.) We decide to skip this detail for now, as it isn't critical to the understanding.

Enclosing_Subpool should be take Acc.

Strong_Reference probably is controlled; it needs to be defined to "needs finalization".

Tucker thought he had a rule to prevent assigning these into access types with a different storage pool manager. But that would appear to disallow converting to an anonymous access parameter.

Gary, Randy, and Steve all are interested in conversion to anonymous with a default pool. There needs to be some sort of accessibility check at such a conversion. That works for access discriminants and parameters. Components we don't want; what about anonymous objects (which have a newly defined dynamic check)?

For statically deeper than, you could assume it is local. More confusion ensues. Steve and Tuck will take this off-line.

Do people understand the proposal? Several people say it is reasonably understandable. Ed worries that it has a number of pitfalls; you would have little idea where the reclamation occurs.

Tucker notes that the weak references could be carved out to simplify this.

The Implementations package seems weird. Tucker says that he tried to separate the stuff the user of subpools needs to know compared to what the implementor of the subpool needs. Randy says that is already mixed, because the end-user doesn't care about the storage pool details.

John says this seems useful and he would like to see further development. Erhard is mostly interested in how the user of the subpools would use it, as opposed to the creator of the subpool manager. Gary says that it seems interesting, but it is hard to understand.

Tucker gave a presentation at Ada Europe, and there was some interest in this idea.

Keep it alive: 9-0-0.

Tucker will try to separate the weak references into a separable part.

Action item to everyone to look at this and see how (or if) they might use it.

AI05-0113-1/04 Conflicting external tags and other tag issues

The 13.3(76) rule doesn't exempt multiple generic instances; they are "separate declarations". Thus it will not be possible to specify an external tag in a generic unit if there is more than one instance. This seems OK – multiple uses of the same tag is a mess. if there is a problem in practice, we would be better off lifting the requirement that the external tag string be static (then the name could depend on a parameter).

There is a duplicate AARM note in the !wording section.

Approve AI with changes: 6-0-3.

AI05-0117-1/00 Memory barriers and Volatile objects

Alan suggests a new pragma that is weaker than Volatile so that it can be more efficient. He suggests "No_Reordering". No reordering means that the CPU has to talk to the cache in the defined order; it doesn't guarantee cache consistency.

Steve says that No_Reorder would also allow removing of dead reads (which Volatile does not). Gary suggests naming the pragma and property "Coherent", which sounds better.

Volatile would imply Coherent (just like Atomic implies Volatile).

Alan will update the AI to propose this pragma.

AI05-0124-1/01 Where is the elaboration check suppressed?

Randy has written this up suggesting that there are different implementation models, and we don't really want to force compilers to change whatever they have been doing.

The group agrees with this position. We could define it to be on the declaration, but that seems to be a little benefit and possibly significant work for implementations.

This should be a !ramification. We don't want an AARM note, as it is more likely to introduce new problems.

Approve AI with changes: 10-0-0.

AI05-0127-1/02 Adding Locale Capabilities

Brad says that he modeled this on the Java packages; they use a two character country codes/language codes. There is now a standard for 3 character language codes. Brad also notes that GTKAda has bindings to locales.

Brad notes that he had to abstract this away as Windows/Unix does this somewhat differently.

All GNAT does is support file names in Unicode.

There is some support for having a way to query the locale. Tucker suggests that locales be defined as strings, so we don't tie the language to a particular standard. (Then 2 or 3 or 10 characters will work).

The predefined ones would probably be a political problem, we'd rather not have any. Brad says those were the ones Java defined. Let them have the political problem.

Tucker thinks this should be in Interfaces: Interfaces.Locales. Steve thinks that System might be more appropriate. So have System.Locales.

Drop all of the calendar stuff, and so on; just one package.

Brad will develop a draft on this basis.

Keep alive: 5-3-2.

AI05-0135-1/05 "Integrated" nested packages

This is an old alternative on the list of AIs expected to be voted no action. See AI05-0074-1.

No action: 9-0-0.

AI05-0135-2/02 "Integrated" nested packages

The idea basically works. But Steve says that this would be a terrible idea if it didn't fix the deferred instantiation problem. A number of other people concur. That suggests that we *still* don't have the right solution.

Tucker says that maybe we need a formal incomplete type, since we have allowed using these in more places. It wouldn't require a frozen type. No objects or components could be used, of course.

This would work well for signature packages (like the iterator generic), but it wouldn't help the containers.

Steve will make an attempt to write up the formal incomplete type idea, with help from Ed.

Tucker moves no action, and Ed seconds.

No Action: 5-1-4. We decide that this vote is inconclusive. Keep it alive until the phone meeting: 5-2-3.

AI05-0139-1/02 User-defined iterators

This is an old alternative on the list of AIs expected to be voted no action. See AI05-0074-1.

No action: 9-0-0.

AI05-0139-2/04 Syntactic sugar for accessors, containers, and iterators

Tucker explains the Reference interface.

Steve asks what happens if there is a component Q of the object and of the type designated by the discriminant: P.Q and P.D.**all**.Q. Tucker says that is ambiguous.

Turning to indexing. Tucker has proposed a function designator "()". The problem is that we need two functions, one for read-only access to an element; one for read-write access to an element. These are homographs by the current rules.

This causes a number of weird cases: we would have to allow the declaration of homographs in this case, and modify resolution in some way. You could choose between the functions based on whether the prefix is a constant view or not. Or you could base it on the context; but that doesn't work when the function is renamed. It looks like a mess.

Randy suggested (in e-mail before the meeting) using aspects for these routines instead. Tucker presents this idea to the group. There are two aspects (one for read-only access, and one for read-write access). The aspects would work on all types; for an array it would just be ambiguous if specified. The idea is that the aspect function name is an overloaded name (that provides multiple index types if necessary).

Tucker then explains the iterator proposal, and finally the default iterator.

Matt asks how you tell whether you want the element to be a constant. It would be determined from whether the prefix is a constant or variable view; the programmer couldn't ask for a read-only element from a variable container.

Bob asks whether he is the only one that finds "of" in the for loop syntax distasteful. Ed says "Yes" drawing laughs (but no one disagrees).

Erhard wonders why we don't make this implicitly parallel. It would be weird to have this implicitly parallel when regular iterators are not. Tucker notes that parallel loops would be very desired for arrays, that would have to be done if we were to add some form of parallel loops.

Steve notes that a parallel loop body would require some hairy additional rules to deal with when exceptions are raised.

Tucker will provide wording for these constructs.

Matt says that STL uses "Forward_Iterator". That seems to be an improvement, replace the "Basic_Iterator" interface by that name.

We briefly talk about what to do when the key doesn't exist in a map. We decide to defer that.

Approve intent: 10-0-1.

AI05-0141-1/01 User-dereferencing in Storage Pools
AI05-0142-1/01 Variable function results
AI05-0142-2/01 Limited access types
AI05-0142-3/02 Accessors for Ada.Containers

These all were on the list of AIs expected to be voted no action. See AI05-0074-1.

No action on all of these: 9-0-0.

AI05-0142-4/05 Explicitly aliased parameters and accessors for Ada.Containers

Why is this needed? It would seem that access parameters could do the job. Randy points out that this gives a static guarantee (via a check at the call site which can fail only in rare cases); if we use access parameters, there would be a runtime accessibility check at the point of the return statement.

Bob says that he very recently had a request for such a feature.

Tucker notes that this also allows pass-by-reference of elementary objects, that would be good for atomic objects.

Steve worries that the wording of C.6(12) doesn't work for a by-reference elementary object. That could cause an atomic object to be accessed by code that doesn't know it is atomic. That needs to be covered in C.6(12). Tucker suggests just dropping the word "type". Bob says it is easy, and he will do it if the editor has trouble.

Steve wonders if there is any problem with dispatching calls. No, because both ends of the check (the parameter check and the return object check) are based on the call site; it doesn't matter where the function body is defined.

Split the container rules into a separate AI that we deal with separately.

Under 6.4.1(6): "apply to all calls" should be "apply to procedure or entry calls".

In a function call, the accessibility level of the actual object for each explicitly aliased parameter shall not be statically deeper than {the} accessibility level of the master of the function result.

Under 6.4.1(15): In a function call, for each explicitly aliased parameter, a check is made that the [accessibility level of the] master of the actual object is the same as or includes that of the master of the function result.

That doesn't quite work. We need to recraft this. Erhard complains that the accessibility of an object is always dynamic, and is always the master of the ultimate actual, which is not what we want. Tucker says that tagged types operate this way and that isn't a problem. But this wording isn't the same as that. Tucker and Randy will try to craft better wording.

Approve intent: 6-1-4. Erhard hates adding more references into the middle of aliased arrays.

AI05-0144-1/03 Detecting dangerous order dependencies

This is an old alternative on the list of AIs expected to be voted no action. See AI05-0074-1.

No action: 9-0-0.

AI05-0144-2/03 Detecting dangerous order dependencies

Ed reports that the implementation effort is not that difficult, and the cost is quadratic in the number of actuals. Someone points out that the cost isn't really quadratic; it's really (actuals * function actuals that are **in out** or **out**) – that's likely to be much smaller (often 0).

John "dependencies" should be "dependences".

Wording: "…name N [of an] that is…"

Steve worries that "denoted by S1" in the wording for slices makes it not apply to "A1(1..10)" (as there is no subtype here to be denoted – this is a range). The wording should say something else. Steve will take an action item to figure out appropriate wording.

Tucker reiterated that he feels very strongly about this. Bob notes that he is in favor of anything that makes Tucker happy (drawing some chuckles).

Approve AI with changes: 5-0-4.

Erhard reports on a study of existing Ada code. He notes that Cooper-Kennedy aliasing detection algorithms found 43 occurrences of aliasing in 500,000 lines of Ada code; maybe a dozen were really bad. So he thinks this check is pretty useless. Tucker notes that there couldn't have been any **in out** parameters for functions in that code, so it cannot prove or disprove anything about that feature.

Tucker says that it is not a problem to implement in practice, based on similar rules in their compiler. Steve says he's mostly worried about the size of the language wording.

AI05-0145-1/02 Pre- and Postconditions

This is an old alternative on the list of AIs expected to be voted no action. See AI05-0074-1.

No action: 9-0-0.

AI05-0145-2/05 Pre- and Postconditions

Erhard asks whether the state of a protected object is visible. No. Depending on the state of the object would be a race condition, because the object is being modified by other callers. That would be a bad idea in general (not just for a pre or post condition), since the caller cannot depend on the values of the object: they might be changed.

X'Old "In particular, if X'Old appears in a postcondition expression, and postconditions are enabled," replace, "For each X'Old in a postcondition expression that is enabled,"

The last line of the Dynamic Semantics needs to define that pre and postconditions are enabled if the Assertion_Policy is Check.

Dynamic Semantics:

Why is the protected action allowed to be started? So the precondition can be evaluated within a subprogram. But that is not allowed for an entry call, so that implementation isn't likely to be used.

The user wouldn't know if they can use blocking operations here. Blocking operations shouldn't be allowed in any pre or postcondition (we don't want "hidden" blocking operations). Add a bounded error to that effect.

Wording fixes for the first paragraph of the Dynamic Semantics:

"If and only if all the precondition expressions evaluate to False, [then] Ada.Assertions.Assertion_Error is raised."

"For {a task or protected}[an] entry call, the check is performed prior to checking whether the entry is open."

"The order of performing the checks is not specified, and if any of the precondition expressions evaluate to True, it is not specified whether the other precondition expressions are [checked]{evaluated}.

"This {begins with}[consists] of the evaluation of the precondition expressions that apply to the subprogram or entry."

It is possible that something might happen between the evaluation of the precondition and the actual call, so if something can change, it might no longer be true in the call. A note to this effect would be nice.

For F'Result:

"For a prefix F that denotes a function {declaration}, the following attribute is defined:"

"{Within a postcondition expression for function F,} denotes…" "[In the former case]{For a controlling result}," "[In the latter case]{For a controlling access result},"

John, notes a glitch in the !proposal: "(that is, all of the postconditions must be satisfied for the overall postcondition [is]{to be} satisfied).

Approve AI with changes to be reviewed at the April phone meeting and pending completion of AI05-0183-1: 11-0-0.

Steve asks if the type of the Old attribute is well-defined. We usually say the type, so we need to add wording to define that.

Do we use the context for resolving the prefix? Yes. Tucker says the wording should be just like parens. Steve points out that there is no wording for parens, so copying that will be hard.

Tucker suggests for 'Old "If the attribute reference has an expected type or shall resolve to a given type, the same applies to the prefix; otherwise the prefix shall be resolved independent of context." [Editor's note: This needs to be a separate Name Resolution Rule, based on the precedent of 'Access in 3.10.2.]

AI05-0147-1/09 Conditional expressions

The syntax rules are just too long for the AARM. It should be in the AI discussion somewhere. Randy would prefer to shorten this to a short English description. Others would like a short link to the AI. Randy will craft this.

"If the expected type of a conditional_expression is any type in a class of types (instead of a particular type), all *dependent*_expressions of the conditional_expression shall have the same type."

This rule is considered weird; several people complain that it shouldn't apply to universal types. Randy tries to explain the intent, that it applies only in the cases where "any <some kind of> type" is expected. Hardly anyone else reads the wording that way, so it doesn't work. It's not clear to everyone that we even want such a rule.

We then get into a lengthy discussion of what we really want. This is not obvious. One problem that shows up is that a type conversion from a conditional expression has problems if the type of the *dependent*_expressions is not the same. Many rules depend on the operand type, and it wouldn't be determinable in that case. We don't want to have to change the rules for type conversion checking to apply individually to each dependent_expression.

Tucker suggests that the type of the operand of a type conversion that is a conditional expression, imposes the type on the conditional expression. That would ensure that no checks need to be made on the operands.

Steve notes that it would be nice if the entire expression had a type.

Randy points out that doesn't work well for class-wide types. We also need to remember anonymous access types, as they surely have different types for every declaration.

Randy shows the original example that drove him to allow different types in most circumstances:

```
function F (...) return T'Class is
begin
   return (if X then A else B);

   if X then
      return A;
   else
      return B;
   end if;

end F;
```

Here A and B have different types covered by T'Class. It would be odd if this didn't work (the regular return statements are both legal). This suggests that we should adopt a principle is that if you can write something as an **if** statement, it should be writable as a **if** expression as well. The reverse is not necessarily true.

Do we need `T'Class(if X then A else B)`? You can't write exactly that as an **if** statement (the type conversion would have to be distributed, and that would work). So we conclude that we don't need to support this if it causes trouble.

The problem case with the rules as written is that something like:

```
NN : constant := (if X then XX else 0);
```

is illegal if XX is a static constant (as opposed to a named number). But both XX and 0 are legal definitions for a named number.

This is a common useful case; if it is illegal we will have griping. Note that it (sort of) violates our principle:

```
if X then
   declare
      NN : constant := XX;
   begin
      ...
   end;
else
   declare
      NN : constant := 0;
   begin
      ...
   end;
end if;
```

This is legal, although it requires duplicating the usage.

Steve will make another attempt at crafting rules.

Approve intent: 9-0-0.

AI05-0150-1/03 Use all type clause

Why don't we change **use type** to do this? There is a compatibility issue, it could cause legal programs to become illegal.

Tucker suggests in addition to compatibility, coding conventions might disallow one and allow the other. John notes that SPARK only allows **use type** (and doesn't allow regular **use** or redefinition of operators).

Straw poll: 7-1-3 in favor of this idea.

Leave the wording as is.

Approve AI: 6-1-3. Bob objects for the reasons noted in his e-mail of June 28, 2009.

AI05-0151-1/06 Allow incomplete types as parameter and result types

Tucker fixed this once, but we decided in a phone call to change it further. But Tucker didn't update it a second time. We don't consider this now.

AI05-0153-1/04 Subtype predicates

Bob notes that whether it is checked on return for an **in out** parameter is implementation-defined, as it is only required for by-copy result assignment. He would like to change this to require a check on all **in out** and **out** parameters on exit. That seems OK.

Randy and Bob argues about predicates on uninitialized objects. Randy wants them checked; Bob wants them checked only if the object is default initialized (record with default expressions, access type). Ed agrees with Randy, and Tucker agrees with Bob.

Bob would like the rule to be like the parameter passing rule (6.4.1(14)). He's not sure about access parameters.

Do validity checks include the predicate? Probably should.

We don't want to allow optional checks. It is too likely that they would cause problems in use.

We do want allow removal of checks for identical subtypes. It is important that this be some sort of permission, as there is no way to tell if a particular predicate will get the same result on successive calls or if the check is in fact still true.

Someone doesn't understand this. So we write an example. Assume T is composite and has a component C (on which the predicate depends). Consider:

```
X, Y : T;

X.C := ...; -- Invalidates predicate of X, no check required.

Y := X;     -- Check could be suppressed here (same subtype), even though it would fail.
```

[The editor had to leave the room at this point and missed about 15 minutes of discussion. Unfortunately, the editor has been unable to find anyone with better notes of this discussion. The missing discussion includes the decision about "plugging the hole" as noted in the vote below; there is no record of the suggested rules or reasoning except for the following sentence (which doesn't come close to plugging the hole).]

Tucker suggests that you have to do the identical check unless the compiler can prove that the object is unmodified since the last check. "Can omit the check if no part of the object has been modified since the most recent check."

Approve intent: 6-1-3. John objects; he doesn't like the hole implied that the predicate can be invalidated by a component assignment. Tucker thinks that this hole is plugged, Randy is dubious.

Bob will take this AI and would like some guidance in e-mail.

AI05-0153-2/05 Set constraints and extended discriminant constraints

We're going to try to use subtype predicates to solve this problem.

No Action: 6-0-5

AI05-0158-1/04 Generalizing membership tests

The resolution of this is a problem; it would not allow many common usages.

We look at the original wording; it starts on the right-hand side. That is somewhat of a fake, the types on both sides participate. Someone says that assignment is that way. Tucker says no, the LHS of an assignment cannot be a literal (or something allowing any type). So assignment is not quite symmetrical.

The problem case is:

```
F in 10 | X
```

"Expected to be any type" would not allow "10" to resolve (it doesn't have a unique type). Someone suggests using "the choices shall resolve to the type of the tested expression". But "Shall resolve to" doesn't allow literals or anonymous access types.

Ed will try again on resolution, working with Steve.

Keep alive: 7-0-2.

AI05-0159-1/04 Queue containers

There is confusion about passing in Synchronized_Queues. That's defining the interfaces.

Change Synchronized_Queues to Synchronized_Queue_Interfaces. The formal package Queues becomes Queue_Interfaces.

The first sentence of each could be improved. For the priority queues, "The type Queue is used to represent {task-safe priority} queues." For regular queues, "The type Queue is used to represent {task-safe} queues.".

John complains that the !summary isn't there. The !proposal should be the !summary. The !proposal should say more about what is proposed "an interface and four implementations of task-safe queues." Then describe the four queues (bounded and unbounded; and priority queues).

The two bounded versions: "The type Queue needs finalization if and only {if} type Element_Type needs finalization."

In the AARM note: "may block" should be "might block".

For A.18.23: " If the queue object has a bounded capacity, and the number of existing elements equals the capacity, then Enqueue blocks until storage becomes available{; otherwise Enqueue does not block}."

Dequeue: "While the queue is empty, then Dequeue blocks until an item becomes available." This isn't liked, leave it as it is.

{In any case, it}[It] then returns a copy of the element at the head of the queue, and removes it from the container.

Do that for Enqueue as well.

Approve AI with changes: 11-0-0.

AI05-0162-1/02 Allow incomplete types to be completed by partial views

We discuss how this impacts AI05-0208-1. Not significantly.

In the changes for 3.10.1(3), "incomplete_type_declaration[full_type_declaration]", there should be a '}" before the square bracket.

A shorter example would be preferred. Steve gives us:

```
type T1;

type T2 (X : access T1) is private;

type T1 (X : access T2) is private;
```

Rewrite the !problem to use this simpler example; move the long one to !example. Say that "the use of an incomplete type prevents the use of information hiding".

Approve AI with changes: 10-0-0.

AI05-0166-1/03 Yield for non-preemptive dispatching

Should this package be Pure? That would be weird. But it should be Preelaborated. The parent package is currently Pure, it needs to be changed to Preelaborated. That's incompatible, but doesn't seem very important.

Typos: "then {the} calling task"

Approve with changes: 9-0-2.

Add an AARM note to explain that when Yield_to_Higher is used in non-Non_Premptive policies, a higher priority task should never exist, so this is a no-op.

AI05-0167-1/02 Managing affinities for programs executing on multiprocessor platforms

Drop "platforms" from the subject (it's just a noise word, and this subject is too long). Just say "multiprocessors".

Alan notes that there have been complaints that the enumeration Policy should be extensible. We did that once, and it was a mess. We're not very interested in doing that again. Implementation_Defined will provide one such policy, Tucker suggests that a child package could be used to provide others.

Tucker wonders about the procedures. They are effectively the same as the existing pragmas.

Ed worries that there would be no implementation. Marte isn't supported anymore. Tucker and Alan say that we are getting out ahead of the curve – this is an area where Ada ought to be a leader, not a follower.

Ed says that the affinities is useful, having different policies for each CPU seems like overkill to him.

Dispatching.Domains in the package name should be Dispatching_Domains (the period should be an underscore). AI05-0171-1 should use the same name.

You can get the effect of different dispatching on different CPUs using priority ranges, and put the appropriate tasks on the CPUs. So there isn't a strong need for this extra functionality.

So we suggest moving the policies package into the !discussion, so we have it for the future. But we won't put it into the language.

Steve worries about what happens if a dispatching_domain goes away (if it is an object declared locally). The wording should handle that. Tucker suggests that the user domain doesn't need to persist.

Probably should require the routines to only be called from the environment task only.

Erhard asks what "current DD" means for Set_CPU. That's the DD assigned to the task. Tasks always belong to a DD. Drop "current" here, as you can't change it once assigned.

Approve intent of AI: 9-0-2.

AI05-0168-1/03 Extended suspension objects

Steve says that the Verrazano Narrows bridge is an "extended suspension object". <Grin>

Ed asks why we need a separate package for this single subprogram. We don't want to drag in Ada.Real_Time into programs using other suspension objects.

Steve asks what happens for non-EDF policies. What is a deadline in that case? All tasks have a deadline, the policy might just ignore it.

Approve AI: 11-0-0.

After the meeting, it was noted that the last paragraph of the new wording belongs in the Dynamic Semantics section after D.10(10). (Else it would come before the description of the other subprograms defined here.) It also needs the wording from D.10(10) about it being a "potentially blocking operation".

AI05-0169-1/02 Group budgets for multiprocessors

People are confused as to why there is no connection between Dispatching_Domains and Group_Budgets.

Alan says that it isn't well-defined what it means for a set of processors; interrupting many tasks at once when the budget expires is an issue.

Ed wonders why a domain that contains multiple CPUs would ever care about one CPU.

Tucker suggests 1 should be CPU'First.

There is a potential inconsistency here, because we are constraining implementations in ways that were previously allowed. If a program depends on some behavior no longer allowed, it could fail. That should be mentioned in the ! discussion.

Approve AI with changes (pending completion of AI05-0171-1): 9-0-2.

AI05-0170-1/02 Monitoring the time spent in Interrupt Handlers

Tucker would prefer that the package is always defined, and returns zero if not implemented. Alan worries that it wouldn't be possible to tell the difference between no time used and not implemented. It is suggested to add a Boolean to say whether this monitor is implemented.

Randy asks what happens if interrupts are accounted, but not separated by Id. That seems as if it might happen. We suggest adding a total interrupt time function. And a Boolean for that.

Should this be in the same package? Tucker says that we don't want to add a with clause to the existing package. Perhaps the Booleans and the total interrupt time could go into the main package, and the subpackage would only contain the by-id version.

Clock_for_Interrupts in the main package.

We don't need the implementation advice, as the Booleans will provide sufficient incentive. So remove it.

Approve intent: 8-0-3.

AI05-0171-1/03 Ravenscar Profile for Multiprocessor platforms

The subject should be changed to include pragma CPU. "Pragma CPU and Ravenscar Profile"

Jean-Pierre asks what happens if the CPU pragma occurs in a task type. The model of the CPU pragma is the same as Priority. So the argument can be a discriminant in a task type.

Resolution should say that the expected type is System.Multiprocessor.CPU_Range. The dynamic semantics should say that you get an exception if it is not in the range System.Multiprocessor.CPU and <= Number_of_CPUs.

Tucker thinks that maybe zero should mean run anywhere. If you have a task type with a CPU discriminant, you might want to specify that you don't care where it runs.

Add `Any_CPU : constant CPU := 0;` or `Not_A_Single_CPU : constant CPU_Range := 0;`

Non_Specific_CPU is also suggested. Alan will pick a name.

There needs to be some words that explain what it means if the pragma appears in the main subprogram.

"Within a given partition, each call on Number_of_CPUs will return the same value." The package shouldn't be Pure because this requires a dynamic query to the OS; the package should be Preelaborate.

There should be a cross-reference to the AI that defines "System.Multiprocessors.Dispatching_Policies".

The type of the discriminant in the example should be CPU_Range.

Steve notes that the last two sentences of the Implementation Advice aren't technically advice. For the last sentence, either reword it with "should" or move it somewhere else. [The penultimate sentence includes the word "should" and appears to be proper advice to me, not sure what Steve's complaint was. - Editor]

[Editor's note: Reviewers of these minutes say they believe that both sentences should be Implementation Requirements rather than Implementation Advice, and thus worded using "shall" and not "should". I don't recall any such discussion at the meeting, but the AI author should carefully consider the intent of these two sentences and locate and word them appropriately when revising the AI.]

Gary asks to get rid of hyphens in multi-processor (Implementation Advice and possibly other places).

Approve intent of AI: 11-0-0.

AI05-0172-1/01 Extension to Ravenscar Profiles

Alan would like to kill this one. He believes that it brings additional non-determinism to the runtime and thinks that the workshop needs to study this further. We are happy to follow his advice.

No Action: 10-0-1.

AI05-0174-1/02 Implement task barriers in Ada

The main point here is to allow using OS facilities.

```
subtype Barrier_Limit is Positive range 1 .. implementation-defined;
```

Users can use Barrier_Limit'Last to find out the maximum number of tasks supported by a synchronous barrier.

Get rid of the System change and with of System.

Steve wonders if you are abort-deferred while waiting here. It's like a rendezvous that has already started, which would be abort-deferred.

Someone suggests that an implementation might want to abort one of these if some other task dies. That wouldn't seem to work very well, because the barrier would wait forever if there aren't enough tasks.

Tucker suggests that this should be implementation-defined if it is aborted after it reaches the barrier and it was waiting on the barrier.

That seems best. We want it to do whatever the underlying OS does.

Pure is OK, there is no state within the package.

Approve AI with changes: 9-0-2.

AI05-0175-1/02 Cyclic fixed point types

John noted that you can't represent reasonable values exactly on a binary machine. For instance, 60 degrees can't be represented. So the whole idea seems untenable.

Ed had shown a possible package implementation of this idea. Do we want to add this package to the Standard? No, it seems weird and an insufficient need. A Gem of the week seems like a better idea.

No Action: 10-0-0.

AI05-0176-1/04 Quantified expressions

The syntax in the AI is wrong, the editor screwed up. The rest is correctly updated.

John suggests using a right arrow rather than a vertical bar. He says SPARK uses a right arrow, and he prefers that.

Quantifier ::= **all** | **some**

Tucker says the grammar should be "*some*_identifier". That is much too language-lawyerly.

We could say "**for not all**". That's not quite right. "**for** 1 …" or "**for** >1 …"

Add a note "Some is not a reserved word". Some is not in bold here.

Quantifier ::= **all** | Some

Straw vote: Vertical bar: 7

Right arrow: 3

Neither: 1

Tucker worries that there is a problem with the vertical bar if we want to extend the membership notation to looping. We had previously talked about that, but didn't do it. Randy says that he doesn't like the bar here as it currently is a list separator (or maybe better described as a set separator), and he doesn't like the right arrow; he would rather have a reserved word.

We again discuss unreserved keywords.

Erhard claims "for some" is weird; Tucker replies by finding a Wikipedia article where backwards E is "pronounced that exists or for some…"

The grammar BNF fix is probably a new kludge: "after a loop_parameter_specification" 1.1.4(12-13).

Approve AI with changes: 6-1-3.

AI05-0179-1/03 Labels at end of a sequence_of_statements

Improve the end of the discussion (some find the examples confusing):

{is illegal,} but

are allowed {by this proposal}.

Should we allow a label by itself? That would require extra rules, because we don't want to allow nothing at all. It's easier to leave this the way it is defined.

It is suggested to reorder the end of the discussion, Bob is sending a suggestion in e-mail.

Approve AI with changes: 10-0-0.

AI05-0186-1/05 Global-in and global-out annotations

We discuss the basic model. This is a conservative compile-time checked annotation.

Bob suggests a default annotation for an entire package. That probably would be in the form of an additional aspect. It would apply to all contained entities that don't have the aspect specified.

Bob and Randy discuss who takes ownership. Randy eventually takes this AI back.

Ed suggests that this is too hard, perhaps you should defer it to Ada 2020. Randy says that is OK by him, but then he will vote against all of the Pre/Post/Predicate things. He notes our charge is to "improve the ability to write and enforce contracts"; he believes that having only run-time enforcement is too late and just adds another hazard from bad preconditions (ones with side effects and/or non-reproducible results). We have pragma Assert for that sort of checking, we don't need anything else [other than a "finally" clause, which we've always needed for many reasons - RLB]. The compiler needs to be able to warn about bad contracts, and it needs to be able to make at least simple analyses of the contracts. He notes that Pascal said something similar in e-mail.

This brings general groans. Tucker notes that the preconditions, et. al. can provide hooks for other tools. Randy says he understands that, but that is too late for agile-style development – no enforcement can be done until the bodies of all involved routines are available. That makes the feedback very delayed. Bob notes that having the possibility of body dependencies mean that contracts have to be continually rechecked: a small change can potentially make previously correct code far away fail its checks. If a usage is proven to meet its contract, that needs to stay true so long as neither the use nor the contract is changed.

Probably we want to simplify this as much as possible - Randy notes that he'd previously tried to add a categorization pragma for this and had gotten no traction. This more complex scheme has additional applicability. Most of the complexity comes from having to handle implicit calls/uses in some way; it wouldn't be correct if those are ignored.

Could this be done by an implementation if we didn't include it in the language? Not currently, the syntax of aspects is not powerful enough to represent these sorts of annotations. Tucker suggests looking at generalizing the syntax for aspects. Perhaps that would provide a hook for future work.

Keep alive for now: 6-2-3. John and Gary vote against.

John says that this seems too complex.

AI05-0187-1/01 Shorthand for assignment with expressions naming target

The e-mail straw poll showed nothing even remotely close to a consensus on this topic.

No Action: 8-0-2.

AI05-0196-1/02 Null exclusion checks for 'out' parameters

The wording was worked out in e-mail.

Approve AI: 10-0-0.

AI05-0203-1/01 A return_subtype_indication cannot denote an abstract type

This couldn't happen in the original Ada 2005, that required static matching with the return type (which could not be abstract). This hole opened up when we changed **return** to "covers".

"we could {use}[declare] a specific abstract of" in discussion.

Add (see 6.5) to the new wording. The new sentence should go after the 5th sentence of 3.9.3(8/3).

Approve AI with changes: 10-0-0.

AI05-0204-1/01 String packages should be Remote_Types

Bob says that the incompatibility is too much. Ed agrees. Nobody speaks in favor of taking the incompatibility (Randy notes that another proposed change [AI05-0206-1] would allow use of these types in the private part of Remote_Types packages, which may be enough.)

No Action: 8-0-2.

AI05-0205-1/01 An extended return statement declares a name usable inside the statement

The new bullet should go after 8.3(17), not 8.3(18). 8.3(18) and 8.3(18.1) are related and shouldn't be separated.

Approve AI with changes: 10-0-0.

AI05-0207-1/01 Access constant is considered for mode conformance

Get rid of the hyphen in "type-conformant". (This is old wording.)

"…they are type[-]{ }conformant, [and] corresponding…"

Approve AI with changes: 9-0-1.

AI05-0208-1/02 Characteristics of incomplete views

The example is legal Ada 83, Ada 2005 made it illegal.

"Declarative list" should be "declaration list" in the new wording.

We probably don't need both halves of the wording; they mean the same thing. But which half do we want?

- it occurs within the scope of *T* after the completion of *T* if *T* is an incomplete view declared by an incomplete_type_declaration.

or

- it occurs within the scope of *T* after the completion of *T* if the completion of *T* is declared immediately within the same declaration list as *T*.

"if" should be "and" here. "Scope of T after the completion of T" is the same as the "Scope of the completion of T".

Try again:

- it occurs within the scope of the completion of *T* and *T* is an incomplete view declared by an incomplete_type_declaration.

or

- it occurs within the scope of the completion of *T* and that completion is declared immediately within the same declaration list as *T*.

We just want to allow incomplete types, not incomplete views from limited with. The first is slightly clearer for that point. So we agree to use the first wording.

The old first bullet ends with "or", so the second bullet needs to as well.

Approve AI with changes: 8-0-2.