Minutes of the 21st ARG Meeting

11-13 December 2003

San Diego, California, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns, Gary Dismukes, Kiyoshi Ishihata, Pascal Leroy, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg (all but Thursday morning), Tucker Taft, Joyce Tokar, and Tullio Vardanega.

Observers: Jim Moore (parts of Thursday), Ben Brosgol (Thursday afternoon), Greg Gicca (Saturday).

Meeting Summary

The meeting convened on 11 December 2003 at 09:00 hours and adjourned at 13:30 hours on 13 December 2003. The meeting was held in conference rooms at the Red Lion Hanalei Hotel, in San Diego, California. The meeting was hosted by Gary Dismukes in conjunction with SIGAda. The meeting covered most of the amendment AIs on the agenda, as well as a few normal AIs.

AI Summary

The following AIs were approved with editorial changes:

AI-217-06/06 Limited with clauses (11-0-1)
AI-230/09 Generalized use of anonymous access types (8-0-4)
AI-231/07 Access-to-constant parameters and null-excluding access subtypes (10-0-1)
AI-254/05 Anonymous access to subprogram types (11-0-2)
AI-296/07 Vector and matrix operations (12-0-1)
AI-301/06 Operations on language-defined string types (9-0-2)
AI-326/05 Tagged incomplete types (10-0-3)
AI-340/02 Mod attribute (11-0-1)
AI-361/01 Raise with message (9-0-2)

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

AI-251/10 Abstract Interfaces to provide multiple inheritance (11-0-2)
AI-279/03 Tag read by T'Class'Input (7-0-4)
AI-307/06 Execution-time clocks (13-0-0)
AI-327/04 Dynamic ceiling priorities (13-0-0)
AI-354/02 Group execution-time budgets (8-0-4)
AI-355/02 Priority specific dispatching including round robin (9-0-4)
AI-364/00 Fixed point multiple/divide (10-0-2)

The following AIs were discussed and require rewriting for further discussion or vote:

AI-188/03 The definition of setting a task base priority is too vague
AI-266-2/05 Task termination procedure
AI-285/05 Support for 16-bit and 32-bit characters
AI-302-01/07 Data structure components for Ada
AI-302-02/02 Container library
AI-315/01 Full support for IEEE 559:1989
AI-318/02 Returning [limited] objects without copying
AI-357/02 Support for deadlines and earliest deadline first
AI-363/01 Eliminating access subtype problems

The following AIs were voted No Action:

AI-365/01 Permissions to create grandchildren of Ada (9-1-1)

Detailed Minutes

Meeting Minutes

The minutes of the 20th ARG meeting were approved by acclamation.

Next Meeting

As decided at the 19th meeting, the next meeting will be hosted by Joyce Tokar in the Phoenix area. The dates are February 27-29, 2004. Several members note conflicts with these dates. Joyce reports that she has already made venue arrangements. She is directed to check whether the dates of March 5-7, 2004 would be available instead. Toward the end of the meeting, she reports that the change is possible, so the 22nd meeting is now scheduled for March 5-7, 2004, at the Courtyard by Marriot Phoenix Airport.

The 23rd meeting will be held in conjunction with Ada Europe in Palma de Mallorca, June 18-20, 2004.

SIGAda 2004 will held be November 14-18, 2004 in Atlanta, Georgia. We'll tentatively plan a meeting for November 19-21, 2004. It may be necessary to have an extra fall meeting in order to keep to the schedule. That would probably be held in September.

Rationale Document

There have been several queries about a Rationale document for the Amendment. Gary Dismukes reports that Ada Core Technologies (ACT) is willing to provide some funding to create a Rationale. John Barnes would be the author.

The document would be aimed at users and potential users of Ada. The idea is to create “buzz” for the Amendment. It wouldn’t include everything in the Amendment (since the Amendment will include a number of fixes to the standard which aren’t very interesting to users); it will be more of a highlights document.

These proposals meet with general approval from the group.

WG9 Actions

WG9 made a resolution suggesting that the ARG can send APIs to WG9 with the suggestion that they be standardized via an alternative process. We’ll discuss that in context of potentially affected AIs.

WG9 also resolved that the day of the week enumeration should start with Monday. That corresponds to the current draft of AI-351. (Note: We did not discuss AI-351 at this meeting.)

WG9 also sent two AIs back to the ARG. Editorial comments on AI-297 had uncovered a number of technical issues with the proposed wording. Randy reported that these were too difficult to fix before WG9 considered the AI. Alan will make the necessary changes. Randy should send Alan a summary of editorial comments on AI-297.

WG9 also returned AI-348. Editorial review on that AI showed a very confused presentation. John and Randy rewrote the AI to fix the problems. However, the fixes were made too late for WG9 to consider them, so AI-348 was sent back. We can reconsider it at our next meeting.

Motions

The ARG thanks our host, Gary Dismukes, for his fine work on the local arrangements for the meeting. Approved by acclamation.

Old Action Items

The old action items were reviewed. Following is a list of items completed (other items remain open):

Steve Baird:

- AI-162
- AI-251
- Create an implementation report for AI-318

John Barnes:

- AI-254
- AI-296 (reopened)

Randy Brukardt:

- AI-214
- AI-301
- AI-326
- AI-340
- AI-351
- Create an AI (AI-362) to study whether additional predefined units should be preelaborable (see AI-266-2 discussion).
- Make an AC (AC-79) to encourage more uniformity in implementations. Some suggestions were:
  - External_Tag
  - Storage_IO of tagged types
  - Array indexed by holey enumeration
  - Static elaboration
  - GNAT attributes and pragmas

Editorial changes only:

- AI-218-3
- AI-230
- AI-252
- AI-296
- AI-317
- AI-348
- AI-353

Alan Burns:

- AI-188
- AI-266-2
- AI-307 (reopened)
- AI-327
- AI-354
- AI-355
- AI-357

Kiyoshi Ishihata:

- Consult with the Japanese SC22 about the acceptability of AI-285.

Pascal Leroy:

- AI-133
- AI-217-6
- AI-285
- AI-315
- Create an AI (AI-360) on No_Nested_Finalization, extending it to cover types in language-defined packages.
- Create an AI (AI-361) on “raise with String”.
- Write an article on the Amendment for the Ada Letters.

Tucker Taft:

- AI-231
- AI-318
- AI-332
- Write a report on access subtypes (with Pascal Leroy). (Now AI-363)

New Action Items

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

Steve Baird:

- AI-51
- AI-109
- AI-251

John Barnes:

Editorial changes only:

- AI-254

Randy Brukardt:

- AI-279
- AI-362
- Make a recommendation to the ARG about the alternatives for AI-302 (with Bob and Tucker).
- Send Alan Burns a summary of the editorial comments on AI-297.
- Update Janus/Ada implementation report for current version of AI-217-6.

Editorial changes only:

- AI-217-6
- AI-230
- AI-231
- AI-296
- AI-301
- AI-326
- AI-340

Alan Burns:

- AI-188

- AI-266-2
- AI-297
- AI-307
- AI-327
- AI-354
- AI-355
- AI-357

Gary Dismukes:

- AI-312
- AI-331

Bob Duff:

- AI-219
- AI-239
- AI-269
- AI-303
- Make a recommendation to the ARG about the alternatives for AI-302 (with Randy and Tucker).
- Be the test creator of last resort.

Pascal Leroy:

- AI-285
- AI-315
- Check that the current wording of AI-326 properly handles all of the examples discussed in previous meetings for AI-217.
- Update Apex implementation report for current version of AI-217-6.

Erhard Ploedereder:

- Create an AI to allow access types with no storage pool in Pure units (see discussion of AI-362) [Assigned AI-366 after the meeting].
- Create an AI to add words to insure that it is clear that B.3 can be used to interface to C++ (see discussion of AI-285).

Jean-Pierre Rosen

- Create an AI on a new restriction No_Obsolescent_Features (see discussion of AI-326).

Ed Schonberg:

- AI-292
- Create an implementation report for AI-318.

Tucker Taft:

- AI-275
- AI-286 (split Assertions_Only into a separate AI).
- AI-288 (split into two AIs: pre/postconditions and invariants).
- AI-290
- AI-295
- AI-318

- AI-325
- AI-329
- AI-344
- AI-345
- AI-363
- AI-364
- Make a recommendation to the ARG about the alternatives for AI-302 (with Bob and Randy).
- Update AdaMagic implementation report for current version of AI-217-6.
- Create an AI on the restriction No_Dependence (see the discussion of AI-353).

Items on hold:

- AI-218-01, AI-218-02 (these unadopted alternatives should be voted No Action)
- AI-250 (AI-345 is thought to be a more promising approach for this problem; it should be voted No Action)
- AI-284 (waiting for more keywords to be defined)
- AI-356 (waiting for AI-357; if AI-357 approved as it currently is, this should be voted No Action)

Detailed Review

The minutes for the detailed review of AIs are divided into existing amendment AIs and non-amendment AIs. The AIs 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 six votes for, one vote against, and two abstentions.

Detailed Review of Amendment AIs

AI-217-6/06 Limited with clauses

Pascal explains the differences from the last version. That mainly is that **limited with** is not allowed in scope of use clauses.

He also added 10.1.4(6). Tucker thinks this rule is too specific. Steve worries that this rule would prevent checking semantic rules as part of syntax (**others** clauses, **in out** for functions). Tucker says it should just say "Before a compilation_unit can be mentioned in a limited_with_clause, an implementation may require that it be legal." Steve says you can't tell whether it is legal until after you've compiled it — which is too late. Tucker points out that order doesn't matter for this rule.

Erhard says that it can be circular whether A and B are illegal. Tucker says that it is just is a permission, there is no requirement to check that one or both of the units actually is illegal.

Tucker suggests a simpler change: just add "or before it can be mentioned in a limited_with_clause" to 10.1.4(6) (and remove the current change).

Paragraph 10.1.1(2) needs to be phrased in terms of incomplete views (see AI-326). Phrasing it in terms of incomplete_type_declaration is problematic in the case of two-part types (incomplete types, private types) because then the limited view would end up with two incomplete_type_declarations.

Approve AI with changes: 11-0-1.

AI-230/09 Generalized use of anonymous access types

There is no need to change 3.4.1(3): the phrase “full type definition” in the current RM does cover an access_definition (see 3.2.1(8)). (Don’t be confused with “full_type_declaration”, which doesn’t include access_definition; subtle is the language...)

We look at all of the other changes, and decide they are fine.

Approve AI with change: 8-0-4.

AI-231/06 Access-to-constant parameters and null-excluding access subtypes

Delete 4.6(49), then Add the listed text to the end of 4.6(51).

We decide that Tucker never updated the AI; this was Randy’s quick hack to reconcile AI-230. Tucker needs to do what was in the minutes.

AI-231/07 Access-to-constant parameters and null-excluding access subtypes

Tucker has updated this following the minutes, other than the generic section.

AI-254 will need to change 4.9.1(2) to handle static matching.

Tucker has concluded that null exclusion has to match for generic formals. It would be difficult to write a generic that would work for both null-excluding subtypes and normal ones. Most existing generics would not work properly with null-excluding subtypes, so it is better to give a compile-time error on these.

Tucker also added null exclusion to formal parameters and discriminants.

Real wording is still missing for 8.5.1(4) and 8.6(25).

Jean-Pierre doesn’t like the fact that a controlling parameter is automatically null excluding. That’s ugly, but it doesn’t seem that we can do anything else. Requiring **not null** would be better, but we can’t do that for compatibility reasons. There probably should be a note to this effect.

Approve AI with changes: 10-0-1 (Tucker will write the missing wording soon).

AI-251/10 Abstract Interfaces to provide multiple inheritance

Steve reports that the changes are mostly in overload resolution. He also checked and corrected the syntax.

Erhard wonders why an interface is not a “derived type”. One problem is with generic matching. Interfaces are “derived from” to get inheritance; but are not a “derived type”. Tucker and Erhard find that confusing. Randy says that the model is that interfaces are not derived, you always get new ones. The only reason for saying “derived from” is to get inheritance of operation.

Interface types have no particular parent type; the semantics stay the same no matter what the order. That’s different from derived types.

Randy worries that if we change it to be a derived type, then we will have to revisit the syntax, and we’ll be going around in circles. Still, the group concludes that the notion that a type is “derived from” without being a “derived type” is too weird, and Steve is directed to try to eliminate this oddity. A quick search through the RM seems to indicate that it would be quite fine to say that interfaces are actually derived types.

12.5.5 ought to say that the actual needs to be an interface type. Something like “The actual shall be an interface type.” Steve thinks that this is not needed, because the “interface types” constitute a class, so the first sentence of 12.5.5 is fine as it is. But it is not a class that is closed under derivation; that bothers people. Pascal thinks that it is important that attribute 'Class mimic the meaning of “class” in the RM, and obviously Some_Interface'Class does not

only cover interface types. Tucker says that the definition of class is closed under derivation. Class is not used for every possible set of types; for instance, there is no class of unconstrained subtypes. So this is not a class, so then the rule is needed.

Steve says that the standard 12.5(6) says that the form of a formal type determines the class of matching types. 12.5(7) then goes on to make that a legality rule "in the class determined for the formal"; each of the cases then defines the "class determined for the formal". Randy points out that this is in square brackets, it is not intended to be normative.

Tucker suggests that the class is "interface types" (closed under derivation), and then you have legality rules to ensure that the actual is an interface type. We also probably need something in 12.5(6) to specify "the class determined…". Tucker would just as soon leave "the class determined" out, and have it determined by the other legality rules. John says that all of the other generic formal types say something about "the class determined for", interfaces can't be different. Unless we take all of the other ones out, which would be too big a change for no reason.

Erhard notes that 12.5(6) only allows a single ancestor, so we need to fix it. Tucker writes a suggested rewording:

"For a formal_derived_type_definition or a formal_interface_type_definition the class is the derivation class rooted at the ancestor type, or for a type with multiple specified ancestors, the intersection of the derivation classes."

That seems redundant, let's simplify it.

Randy points out that we already say the "class of all array types", which certainly is not closed under derivation. Such classes are very broad, then we tighten up with Legality Rules.

So we eventually conclude that Steve has this wording right. Tucker wants a "To be honest -- The class of all interface is either not a true class, or includes non-interface types. We're not going to say which, the legality rules insure the right result either way."

The title of 12.5.1 mentioned in the wording is wrong.

The first legality rule in 3.9.4 is static semantics. Static semantics and legality can go in either order in the standard.

Explain or eliminate the distinction between "derived type" and "derived from".

"Ultimate ancestor" is used rarely, and only access types seem to care (and they can't have multiple ancestors.) So we should allow multiple ultimate ancestors; add a note saying that "the ultimate ancestor" is well-defined for untagged types. 3.4.1(10) would change "The ancestor" to "An ancestor". Then add "For an untagged type, there is only one ultimate ancestor, and it is designated as *the ultimate ancestor* of the type."

Approve intent of AI: 11-0-2.

Steve will update the AI.

AI-254/05 Anonymous access to subprogram types

John explains that the AI was reconciled with AI-230/1. He has a couple of issues.

Should **protected** be allowed here? For non-closure ones, the accessibility rules are the same as named routines. No one can think of a problem with the closure case. So allow **protected**.

Ed notes that aggregates of one element are still not allowed, fix the initialization of A_Record in !proposal.

Second paragraph of !problem, third line, "by" should be "be".

What is the convention of an anonymous access type? It has the convention of the enclosing construct (except for rename, which takes it from the actual). But not for **protected** (it's **protected** in that case).

Approve AI with changes: 11-0-2.

John will make the changes and provide them to Randy ASAP.

AI-266-2/05 Task termination procedure

Alan explains the changes to the AI. Wording has been added.

Where should this new clause be added? This is general functionality, it isn't restricted to real-time uses. That suggests it should go in section 9, but it can't be there because it uses Task_Id. So it should go in Annex C (probably as C.7.3).

Changed the name of Get_My_Default_Handler to Get_Own_Default_Handler.

Jean-Pierre notes that default parameter value for type Handler is useless; Handler routines are called from the runtime system. The default expression just makes a Handler more work to write.

Typo in the second paragraph of dynamic semantics: "is already" should be "has already".

The third paragraph is overspecified. Just say: "When a task terminates, a termination handler is determined as follows:"

Jean-Pierre worries about using a local protected object for a Handler. That would fail the accessibility check.

Jean-Pierre notes that the text in the third paragraph that says "no exception id is given" should say "Null_Occurrence is passed to the handler", since the parameter is an Exception_Occurrence, not an Exception_Id. Similarly, the last sentence should say the "associated exception occurrence is passed" (not "set").

Erhard: there is a spelling error in the fifth paragraph of dynamic semantics: "propogated" should be "propagated".

Erhard asks what is the effect of calling Set_Handler with **null**? Alan says the default handler comes back. It would be nice if the wording said that. For Set_Dependents_Default_Handler, it removes the default. The model should be described more precisely.

Why are Get_Own_Default_Handler and Get_Handler procedures with one **out** parameter? It seems like these ought to be functions; make them functions.

The model probably should be described operationally; each task has two handlers (a specific one, and a default one). Get_Enclosing_Default_Handler is suggested. Get_Specific_Handler is also suggested.

In the discussion, change "…for {its}[all] dependent tasks."

No vote, intent approved last time. Alan will update the wording (especially the model) as described above.

AI-285/05 Support for 16-bit and 32-bit characters

Pascal reports that he tried to use ISO 10646. But it has removed most Unicode attributes to notes. So he has referenced the notes in ISO 10646.

What is TR 10176? It is a standard for writing standards, and it completely incompatible with Ada (i.e. "compile-time error" is supposed to be called "exception"). So we don't want to use that. Kiyoshi says that this report gives the international agreement for character sets and it is planning to be updated. But it doesn't cover things like case conversion or normalization forms. Pascal thinks that this is pointless without those things.

Kiyoshi suggests separating Wide_Wide_Character and identifiers. That could be done consistently. Pascal says that the big win is in the identifiers. Jean-Pierre wonders if 10646 is a subset of Unicode — yes, it is.

We decide to discuss Wide_Wide_ separately. No one has any comments on it, so there is no discussion.

We take a straw poll on approving that portion of the AI: 12-0-1.

Turning to identifiers. The possibilities are:

- Do nothing with identifier (just do Wide_Wide_); or
- Adopt the 10176 report (but that does not have case conversion); or
- Base it on 10646 directly (as Pascal has).

We discuss open issue #2. Should we allow non-latin digits in numbers? There doesn't seem to be any need to go that far. Pascal says that he doesn't feel strongly, so he withdraws the idea.

Open issue #3: Should we require normalization? Erhard says that this is a lose-lose situation. If you do impose normalization, tools like grep don't work on Ada text. If you don't impose it, code that looks completely identical (and is semantically identical) is not the same.

Tucker suggests that a compiler could report that the there is a mismatch but report that the error is a normalization one. Erhard thinks that's a problem for the user, because they don't have any way to fix that. Unicode editors typically produce only one of the possible forms, and don't provide any way to change to a different form. Thus, program text created on one editor and later edited with a different one may not work, even though the text looks identical.

Pascal notes that there are different normalization forms for case-sensitive and case-insensitive programming languages. That surprises many.

Ed asks what other languages do. Pascal reports that C is doing the same work with exactly the same problems. They haven't reached a firm conclusion yet; their timeframe is similar to ours.

Tucker says maybe we should just use Implementation Advice that a mode ought to be provided to do normalization. Ed notes that there is another rule that gives an escape hatch: 2.3(6). Tucker thinks that this is so dynamic that specifying normalization won't improve portability. It will change often, because the compiler would use whatever normalization is current when they are built; that won't be the same for all compilers. Pascal thinks that normalization is important: the Unicode folks spent a lot of time specifying how programming languages should do normalization, why would we think we understand these issues better?

Straw poll on making normalization implementation advice: 3-1-7. Should normalization be required: 1-7-3.

B.3(39): char16_t is there twice; it should be char32_t, and should return Wide_Wide_Character. That is the current draft of the C standard. How far are they behind us? They're about parallel with us. We'll have to be careful to update this package if needed for changes in the C standard.

Back to 10176. Should we allow digit-like as the first character of identifiers? Unicode is very clear, and says "no". We have to reference 10646 for the upper case; the open question is to reference 10176 to composition of identifiers or 10646+Unicode recommendation. WG9 needs to decide that; both have problems from our perspective.

Erhard wonders if Interfaces.C still works with the C subset of C++. He mentions that politically it would be sensible to add words that C++ can be accessed through that package. Erhard takes an action item to add such words to B.3.

Keep it alive: 10-0-1.

Pascal will update the AI and solicit input from WG9.

AI-296/07 Vector and matrix operations

The changes were to remove Eigen child packages, and add an accuracy requirement for inner product.

Pascal: Machine_Radix, Machine_Mantissa should have the formal type name Real. "Real'Machine_Radix". (Same in the complex part.)

Jean-Pierre: Why is there a private part in Generic_Real_Arrays? There are no private types. Remove it.

The documentation requirements should be in a Documentation Requirements section, not in a sentence that says "shall document".

Jean-Pierre wonders if the “corresponding operation” is a problem when an operation reemerges. No, this refers to the formal type. Similar wording is used for Generic_Complex_Operations.

Approve AI with changes: 12-0-1.

AI-301/06 Operations on language-defined string types

Replace the !proposal with (See wording.); it's redundant otherwise.

A.10.7(17): “lower bound 1” should say “lower bound of 1”.

First example has an extra “declare”.

Immediately following that example “quite some” should say “quite a lot of”

A.11(3) “exception” should be “except”.

At the beginning of the discussion, “convinient” is misspelled.

Approve AI with changes: 9-0-2.

AI-302-01/07 Data structure components for Ada
AI-302-02/02 Container library

WG9 has given us a tool to send this to an alternative process (an International Workshop Agreement—IWA). Tucker worries that sending this AI back to WG9 for discussion in an IWA will send the wrong message to groups that we requested APIs from.

Tucker thinks that we will need ARG review of these in any case. Randy wonders if there is any use to secondary standards; the previous track record is not good for the use of these. Erhard thinks that having implementations available in a standard way has value; he suggests having a standard place with a control of the namespace.

Tucker suggests a small review subcommittee of the ARG to look at these AIs proposals.

Joyce says that IRTAW worked through the process without bothering the ARG. Randy thinks that’s unfair to the containers working groups, which followed the process to roughly the same point. The real-time AIs were far from perfect when they were submitted.

Pascal doesn’t think that we can make progress at the ARG level. Tucker says that the subcommittee would put together a recommendation. Ed notes that there is a high level of interest from users on this, from a purely political basis (this is something that it is perceived that Ada is missing).

Is this too big to be in the standard? It can be all written in Ada. But other languages have this sort of functionality in their standards, for instance the C++ STL.

By February 1st, the subcommittee should provide a short report that could be used as feedback to the authors of the proposals, and make a recommendation to the ARG. If the proposals aren't mature enough, the recommendation could be that the proposals be used as the starting point for an IWA. Randy and Tucker volunteer, and Tucker volunteers Bob.

Erhard points out that AI-302-1 doesn’t provide a global overview of the libraries. He would like more a roadmap of what is being provided.

Pascal notes that some of the interfaces in 302-2 are strange. In particular, the way iteration is done.

It is suggested that the best ideas from both AIs be extracted.

AI-307/06 Execution-time clocks

Alan explains the changes from the previous approved version of the AI. IRTAW 12 determined that it is important to be able to wait for one or more timers to expire. Thus the package was restructured.

Pascal notes that the dynamic semantics paragraph that starts "For any of the Arm procedures…", should say "When a Timer expires…". This paragraph doesn't have anything to do with Arm.

Pascal comments that the sentence "Package Execution_Time contains a type called Timer, which represents a software object that is …" is overly wordy. Just say "Type Timer represents a software object that is …".

Tucker comments on the description of Disarm. "The Disarm procedure sets the timer to the disarmed state. In this state no timer expirations occur.". The latter sentence could be read to say that no timers expire. It would be better to define "disarmed state" early (it's the default state), and then put nothing here.

Jean-Pierre thinks that there is a race condition between Time_Remaining and Disarm, if this is used for a budget. No, there is no problem. If the task itself is querying Time_Remaining, the clock stops when it is preempted, so it doesn't matter. And if it is a controlling task, that task had better be running at a higher priority, or all heck will break lose.

Pascal asks about the seventh paragraph of dynamic semantics. Usually, exceptions are discussed last. Swap the last two sentences.

Erhard asks what happens if the handler is **null**. A **not null** constraint should be used to prevent that. (If those are not in the final standard, we'll have to revisit this.)

Erhard notes that there are a bunch of operators defined, but no description of what they do. There should be a sentence saying that they have the usual capabilities. Take the one from Real_Time.

Erhard wonders if the CPU_Time stuff is useful enough to split out separately. The event stuff would then be a child. Tucker says that many more programs will refer to the clock than to the timers based on clock. Plus, many implementations will want to support the CPU_Time without the events. Call one package Ada.Execution_Time (and add a **with** of Ada.Real_Time) and the other Ada.Execution_Time.Timers. That gets general agreement. The events will be D.14.1.

Pascal notes a typo in the first paragraph of Static Semantics: Cpu Time => CPU Time.

Erhard notes another typo in the last line of the wording — "limit is exceed".

Approve intent of AI-13-0-0.

Alan will update the AI.

AI-315/01 Full support for IEEE 559:1989

Pascal explains that this is introducing a new numerics mode which is stricter than "strict". Tucker thinks that this should be called IEC_559 mode; this is an ISO standard. Pascal says that package Interfaces has IEEE_32_Float. No, that's an AARM recommendation (which is not normative, of course). But Apex and GNAT use those names.

Tucker would like to get rid of "Get_" from these names.

Tucker wonders about selecting the mode. Perhaps we need a partition-wide configuration pragma to select the float mode. "pragma Float_Mode (…);"

Ed asks if there is a real gap here. Is there a real problem getting infinities on float math? GNAT doesn't check for float overflow. Apex and Janus definitely do.

Is infinity static? That gets a groan from implementers. That requires lots of messing around with the universal evaluators. It seems necessary. But you could say that NaN and infinity are not static. That would eliminate problems, and also would get rid of static matching. Especially as we don't want a static signaling NaN. Steve says:

Signaling NaN could kill optimization — you can't eliminate exception handlers. But that doesn't seem important enough to change IEEE.

Also, unordered math is fun.

What about the attribute 'Valid? It checks the subtype. It seems to be implementation-defined now. We don't really need to do anything. But some Implementation Advice would be a good idea.

This would be the first full implementation of IEEE math in a high-level language. Seems like a lot of work for bragging rights.

Tucker wonders why this is a lot of work. "not (A = B)" cannot be transformed into "A /= B" — which is a fundamental axiom of optimizers.

Does this matter? Tucker thinks that implementation of infinities and NaNs is necessary. Is there really a demand?

A way out of this would be to give an implementation permission that a comparison containing a NaN provides an unspecified result.

We need to ask implementers for implementation impact of this proposal.

Keep the AI alive: 9-0-4.

Pascal will update the AI and solicit input from implementers.

AI-318/02 Returning [limited] objects without copying

Tucker describes his most recent thinking on this topic (based on the recent e-mail discussion). The existing return-by-reference capability for limited types gets in the way of a proper definition of constructor functions for these types. Also, the contexts where return-by-reference can be used are different than those where other functions can be used. The interesting contexts for new structures would not allow return-by-reference functions. So we need to explicitly identify such functions.

Return-by-reference is not privacy-preserving (the accessibility only applies to "really limited" types). With the explicit declaration, return-by-reference always happens.

This is a compatibility issue, but problems are likely to be rare. It is nice that return-by-reference is no longer based on whether the type is limited, that should eliminate various generic contract model problems. Are there really contract model problems with functions? Tucker replies that there is already a contract violation problem here, it just doesn't show up much. (But he fails to provide an example.) You do really need to know about this property in generics, etc. (so you can check the legality rules on calls) so, the new syntax is part of access-to-subprogram and generic formal. If it wasn't, you'd definitely have contract model problems.

The new return statement is needed to specify the place for build-in-place. We don't require build-in-place for non-limited types, but recommend it. Note that the subtype appearing in the new return statement must statically match the return subtype of the function if the return subtype is constrained.

The return is simplified by not allowing access to the discriminants, etc. of the destination object.

The goal is that the same user model it true for both limited and non-limited; only assignment is different.

Jean-Pierre thinks is hard to explain when to use the new return statement. Tucker says this is a way to tell the compiler that this is the object that you are going to return. You'll get an error if you need it and don't use it. But it can always be used anytime you need to "build up" a result.

Tucker would like to allow "return;" inside of the return block, but no nesting of return statements with results.

Erhard asks about **exit** (or **goto**) in a return statement; what if it exits out of the block. Probably would like this to work like an accept statement, and reuse those rules.

Does an unhandled exception in the statements propagate directly to the caller, or to any handler? If it is any handler, you could re-enter the return statement. That could mean double initialization (including registration for finalization, restarting of tasks, etc.) Steve suggests to remember that and raise Program_Error if another return happens. Tucker will think about this.

What is the accessibility level of the return object? It should be like an anonymous access parameter.

aliased return means that you have to return a global object. Jean-Pierre wonders why that is in this AI. Because it is in the way of what we want to define the meaning of limited functions to be; we want to move it out of the way.

Erhard says that if we need constructors, we should define constructor. Tucker thinks that we already have this, and a new definition is redundant. Pascal says that you wouldn't have any reason to decide between declaring a constructor or a function.

Greg asks if an unconstrained object is returned as an aliased case, and it is returned and an exception occurs, is the space deallocated? If it is aliased, the object already exists, so there is no problem. If it is not aliased, it would be the same as an exception raised in an allocator.

The runtime cost could be reduced by allowing these only for constrained return subtypes. That wouldn't prevent adding more extensive capabilities in the future (which is an important goal).

Ed says that he would probably vote against if it is fully general. Too expensive to implement, especially at the current late date.

Straw poll on keeping it alive 7-0-5. Adding the restriction of limiting to constrained types: 3-1-8.

Tucker will write up the wording, with the both cases described.

AI-326/04 Tagged incomplete types

Randy explains the changes. The wording introduces the missing "incomplete view", and unifies incomplete and partial views around "view of a type".

Tucker suggests adding "considered" to the last line of 3.2(4). (He says Bob wouldn't let him do it last time.) "An incomplete or partial view is considered a composite type."

3.7(1): The description of the change is wrong; the change is in the last sentence of the paragraph.

3.9.2(2) should say "Add to the end…"

3.10.1(3): Remove the "…declares an incomplete view of the type;…" from here; that belongs in paragraph 11 under static semantics.

Tucker wonders if allowing completion by a private type would cause problems in implementations. Randy doesn't think so.; Pascal says that he thinks **limited with** probably is much worse; Gary and Tucker think that such completions might be a problem.

Erhard wonders if it ever would be useful. Tucker doesn't think so. Gary says that you would be able to write a visible discriminant of yourself:

```
type Foo;
type Acc_Foo is access Foo;
type Foo (A : Acc_Foo) is private;
```

It doesn't seem like users have been clamoring for this capability. We take a straw poll. Can an incomplete type be completed by a private type? 0-6-6.

Therefore, revert all of the text to "full_type_declaration". 3.10.1(3) is unchanged.

3.10.1(4): move the second sentence to 3.10.1(11). Change to "full_type_declaration".

Old 3.10.1(9): Jean-Pierre suggests that we get rid of the compatibility clause J.10. It seems really ugly to "open a hole" as Pascal says. Randy points out that it is a compatibility problem both ways: if you change code to use "tagged incomplete", it no longer compiles with Ada 95 compilers.

Pascal suggests a restriction "No_Obsolescent_Features". That seems like a good idea. Jean-Pierre Rosen volunteers to write an AI.

Erhard: add to 3.10.1(5-9): "In addition, {a name that denotes} a tagged…". A similar change should be made in the last bullet.

Third bullet: there is an integration issue here: AI-254 allows access-to-subprograms in access_definition, but AI-326 talks about "the subtype_mark in an access_definition". This phrase doesn't make sense for access-to-subprograms, because there may be many subtype_marks there (the parameters, the return subtype). So during integration a cautious review of the wording will be needed. Randy is directed to keep a list of integration issues somewhere.

The following should be added to 3.10.1(11) as noted above: "An incomplete_type_declaration declares an incomplete view of the type; if it includes the keyword TAGGED, it declares a tagged incomplete view".

Tucker would prefer to put the part of 3.10.1(11) defining the meaning of a limited view in 10.1.1(12) (see AI-217-6). He suggests changing the second bullet to "For each type_declaration in the visible part, an incomplete view of the type is declared. If the type_declaration is tagged, then the view is a tagged incomplete view." This eliminates the direct use of the syntactic form.

Much discussion of the correct model ensues. Randy and Tuck will take this discussion offline tonight, and return with a revised version of the wording tomorrow.

The first bullet of dereference may not be needed. But eliminating it might re-introduce ripple effects. Randy and Tuck will look at that, too.

Other paragraphs are OK.

Change "Non-tagged" to "Untagged" in title of J.11.

AI-326/05 Tagged incomplete types

Tucker rewrote this wording on Thursday night in consultation with Randy, Pascal, and Steve.

He describes the changes. The main change to 3.10.2(5-9) is to separate out formal parameters/results of access-to-subprogram.

Similarly, primitives of the incomplete type have to be where the complete type is declared. That fixes the hole with dispatching. Note that an incomplete type can always be completed with a tagged type, so this hole can happen with any incomplete type. Therefore, the rule has to say "primitive operation", not "dispatching operation".

3.10.1(10) was changed to "Prefixes of a name shall not be of an incomplete view." This works because we know that the formal cannot be incomplete in any problematic cases. (That's why we need the changes to the handling of access-to-subprogram.)

Tucker wonders about assignment. Clearly, incomplete views should be limited. What if one side is complete? Tucker thinks we should just let this work (if it is otherwise legal).

The AARM example for 3.10.1(11) use "at" as a name, but of course that is a reserved word; change the name.

After the AI-217 note, all of the stuff following up to 4.1(9) is leftover junk from the previous version. It should be removed.

Pascal will go back and insure that this wording covers all of the examples from the minutes of previous meetings to insure that they work properly.

Tucker suggests changing the name of the AI to "Incomplete types".

Approve AI with changes: 10-0-3.

AI-327/04 Dynamic ceiling priorities

We review the wording of the AI.

The attribute should be described in static semantics. That's the first three paragraphs. Attribute definitions are indented. The first sentence should start with "Denotes a component of...". Put a period after System.Any_Priority.

The second paragraph is unneeded, 9.5.1 covers it. There could be a note if it is felt to be important.

So, the attribute definition should read: "Denotes a component of the enclosing protected object P. The attribute denotes the priority of P. Reference to such an attribute shall appear only inside the body of P."

[Editor's note: This wording is missing the type of the attribute. It should read "Denotes a component of the enclosing protected object P. This component is of type System.Any_Priority and its value is the priority of P. Reference to this attribute shall appear only inside the body of P."]

What is its initial value? We need to say that "Its initial value is set by pragma Priority or Interrupt_Priority, and can be changed by an assignment." The existing paragraph 3 is unnecessary and should be removed: the meaning of the attribute doesn't depend on the locking policy. (The meaning of a protected object's priority does, but we don't need to say that here.)

Use "If the locking policy Ceiling_Locking is in effect…" to start the fourth paragraph; it's much cleaner and doesn't refer to specific pragmas. (Perhaps there is some other way to set the locking policy in a particular implemention.)

Get rid of the second "If the locking policy..." by saying, "If this new ceiling priority is below….". So that means that paragraphs four and five are combined.

Jean-Pierre thinks that the bounded error occurs too soon. Several people describe the model of ceiling priorities. The "both" case is weird; Alan says that it might happen. But it would be nice to say that "after executing the entry body." The first one should say at "any time prior to executing the entry body, Program_Error is raised in the queued task." Also, the third bullet should say "and then" instead of "and" to make the sequence of events more explicit.

It is suggested to shorten the second bullet: "the entry body executes at the ceiling priority of the protected object;" Alan thinks that "when the entry is open" is needed to prove this doesn't override the regular rules. Someone suggests: "When the entry opens, the entry body executes…" Most people think this is better.

In the third bullet, replace "queued task" by "calling task". Calls are queued, not tasks.

"…of any task whose call is queued…" in paragraph 5.

Tucker wonders if having code in the Metrics is a good idea. Joyce and Alan point at D.12. Sigh — everything in is the Standard somewhere. But it would be good if the code were correct Ada. This looks like a body, but the syntax is that of a specification.

Never say "'Priority"; it is just "Priority" or "attribute Priority" if there is any possibility of confusion.

"There are no semantic dependences on the package Dynamic_Priorities, and occurrences of attribute Priority."

There are typos in !example: "on how" should be "of how". "New value is return" should be "New value is returned". "initialized" should be "initialized".

We don't want to copy the four bullets, having them multiple places. How to share that??

Randy suggests the possibility of defining "a bounded error of a priority change" in D.5(11), then refer to here.. Tucker suggests hoisting this to D.3. Add a bounded error section after D.3(13). Then refer to it here. "If the priority of the PO or task changes…"

Approve intent of the AI: 13-0-0.

Alan will update the wording as suggested and fix the typos.

AI-340/02 Mod attribute

Correct “an value” to “a value” in the !problem.

Change “types T” to “types” in the !proposal.

This attribute goes before 3.5.4(17), not after, because of alphabetical ordering; fix the wording section (the corrigendum section is correct).

Approve AI with changes: 11-0-1.

AI-354/02 Group execution-time budgets

Since we changed the name of AI-307’s package, this package name needs to change too. Make this clause D.14.2.

Type Task_Group should have a different name. Task_Array is agreed on.

Pascal wonders how this package differs from the other package (AI-307). Why can’t you just have a group of one? Alan says that the problem is that a task can be the member of only one group. But you could want one of each kind.

Ed asks what task would use this package. Alan says that it would be a logical master task, usually running at a very high priority. (Not an Ada master.)

Tucker notes that there is no way to find out about budget overrun. Would that be useful?

Erhard asks what happens if the handler is **null**? (This is getting redundant!) The handler parameter should have a **not null** constraint.

Erhard also wants a separate exception for adding and removing from groups. Use Member_Error for this exeception. Pascal would prefer Constraint_Error. Straw poll: Leave it as it is (using Group_Budget_Error): 8; make some change: 2; abstain: 3.

Tucker: The second Is_Member seems to be asking about membership in some default group. This really is “Is member of any group”. Probably it should be renamed. Belongs_To_A_Group is suggested. Is_A_Group_Member is selected.

Erhard wonders about calling Budget_Remaining from a task in the group. Then the budget will continue to drop.if you do that in a loop. He would like some IA to suggest how often the budget is updated. Is a mechanism that only updates the budget on context switches correct? Do you have to do it on every clock tick?

This matters, because we need to know how quickly the handler is triggered. Users will need to know that to be able to use it, and implementers also need to know. Alan says that AI-307 has documentation requirements. We need something like those here, to specify the latency of the handler (or an explicit dependence on 307.)

Jean-Pierre suggests reordering the operations - move Budget_Remaining and Budget_Has_Expired up after Add. Put all of it together. Erhard would prefer ordering as members, budget operations, handlers. Grouping seems most important, the specific grouping is less important.

Pascal says that future tense wording should be removed.

The wording should say “Group_Budget needs finalization.” And nothing beyond that for finalization. Pascal wants this in Static Semantics.

Erhard asks what happens if Min_Handler_Ceiling is violated. Tucker points out that wording should never use "must"; and should use "shall" only for a legality rule. Aside finished, Tucker suggests that this should say "this is the ceiling that will insure that no ceiling violation will occur".

Tucker points out that the example is wishful thinking: you can't index by Task_Id (it is a private type).

Typo in the example: "real_rime".

Make this a sibling of the event package defined in AI-307.

Approve intent of AI: 8-0-4.

Alan will update the AI.

AI-355/02 Priority specific dispatching including round robin

Alan explains the wording (which is new).

The syntax doesn't match the minutes "*policy*_pragma_argument_association", not "parameter". The priority and other parameters should have names. "*first_priority*_expression". Should just use "*policy*_identifier" as in other pragmas.

Shouldn't say anything about the policy-specific parameters in the general. That should be done where the specific policies are defined.

Change "must equal" to "shall equal". Similarly change "has a value greater…" to "shall have a value greater…".

Why can't you specify a range of priorities for some of these, e.g. FIFO_Within_Priorities? It is well-defined as to what it means. Eliminate the restriction.

The name EDF_Within_Priorities seems misleading. It's more like EDF_Across_Priorities.

The Static Semantic rules are really Post-compilation Rules. Certainly we don't say "is rejected"; use "shall". The last paragraph is still a Static_Semantic rule.

Priority_Specific should be defined in D.2.2, and just forward reference to this section. No, that was changed by AI-321. In any case, it's not a Legality rule, it is a Static Semantic rule.

There needs to be some statement saying the arguments of the pragma depend on the policy_parameter. "The syntax rules and interpretation of the pragma arguments depend on the specific *policy*_identifier."

The model is that only one policy can be associated with a priority. Where is the rule that says that? Add "At most one Priority_Policy shall be associated with a single priority in a partition." (Or something like that.)

The second sentence of the introduction should be in Static Semantics.

There shouldn't be specific policies defined with the pragma; do this similarly to the new D.2.2 from AI-321. Tucker suggests that the *policy*_identifiers should be unified with the task dispatching policy. That is, we say that some of those task displaying policy identifiers are allowed here.

Tucker would rather call it "Priority_Specific_Policy". The policy is on the priority level, not the priorities. This gets general agreement; change this.

The third Static Semantics rule is just a Legality rule (not a Post-compilation rule); we reject the pragma, not the partition.

Pascal worries about the use of System in this pragma as it is a configuration pragma. That seems to be an unlikely problem.

Ed says "In Documentation Requirements, the second sentence shall have a verb." Last paragraph of Dynamic Semantics should be generalized to any defined task dispatching policy.

First sentence of dynamic semantics "when … Priority_Specific is in effect…

Jean-Pierre suggests that we don't really need two pragmas here. We could add all of that to Task_Dispatching_Policy. Erhard and Pascal think that the extra documentation is worthwhile. We decide to leave this alone.

Pascal: Why is Round_Robin a child of Real_Time? It should just have a **with** clause, it doesn't have anything to do with measuring time.

Erhard worries that compilers may want to compile-in task dispatching points, and that would cause problems moving tasks between dispatching models. Non-preemptive could have problems that way.

Pascal suggests replacing Real_Time with an empty Dispatching package. This package doesn't seem to be a top-level package. As a matter of fact, something like this was decided at the Sydney meeting, but it has apparently fallen through the cracks. AI authors are invited to RTFM (Read The Friendly Minutes).

The function should be is Is_Round_Robin instead of Round_Robin; it's a query — that was said last time, too.

We probably should move the exception into package Dispatching. Or do we need the exception at all? We could return 0, but that is a strange style. "Priority_Error is raised when applying an operation associated with a particular policy to a priority with some other policy."

Use "An RR task…" instead of "A RR task". But it would be better to make a blanket statement "This describes the characteristics of a round robin task…", and get rid of "RR task" altogether.

Remove "same" from the lead in before the bullets. Humm, no bullets on the bullets; add bullets!

Correct the third bullet: "follows [the] that for"; ({See} D.14).

Tucker: first sentence of the 4th bullet should be a general rule for Priority_Specific. Tucker says that it is the active priority that determines the dispatching policy. Because otherwise if you have a higher active priority, and the base priority is changed into (or out of) a RR level, we are changing to that priority. So this rule should be based on the active priority. Jean-Pierre says that isn't right. There is a bug here in any case, that needs to be fixed.

In regards to the 6th bullet, Tucker thinks it makes more sense to have an infinite quantum and stick with a single scheduling algorithm at a single priority. That gets general agreement.

Jean-Pierre asks what is meant by "next without an inherited priority" in the 5th bullet. This is hard to parse at best. Erhard: get rid of the "the implementation detects that". Then, "When the task exhausts its budget…".

Typo in "Im{p}lementation Permissions". This is not a partition-wide check; it is just a check on the pragma. It would be better to say: "An implementation need not support Round Robin for every priority level."

Implementation Advice: "will" ought to be "should". Tucker says that it really should say "An implementation should use a quantum as close as possible to the value given by the *quantum*_expression." Shouldn't this be a Documentation Requirement? We want the implementer to tell us how close we can get. Alan says that's already there. Then we don't need the Implementation Advice at all.

Your active priority determines the dispatching points. They could differ from those for your base priority. That should be mentioned in the general model.

Randy wonders if any special rules are needed for FIFO_Within_Priorities. No, no special rules are needed.

Approve intent of AI: 9-0-4

Alan will update the AI.

AI-357/02 Support for deadlines and earliest deadline first

Alan tries to explain why deadline scheduling is important. Tucker and others thought that preemption-level locking could be expressed in terms of priorities, and the AI now uses this model.

Rule 4 means that delays that don't delay don't really do anything at all. (If it was running with the shortest deadline, it still would have it and thus would still run.)

Rule 5 isn't right. It should be equivalent to rule 2.

Rule 6 seems wrong because it mixes Lower and Lower+1. A ceiling priority of Lower ought to be prohibited, otherwise the results would be surprising.

Erhard asks about two separate EDF groups. Those are separate, and anything at the higher level will run first. If there is interaction between bands, you get the standard ceiling priority protocol.

Are multiple bands like this useful? Yes, Alan describes a mixed level scheme used to handle overloads. EDF doesn't handle overloads well; the current suggestion is to pull out the hard requirement tasks to fixed priority before they miss their deadlines. These cannot be expressed in any existing programming language (real-time Java does this wrong).

Returning to the rule 6 question. What does "prohibit the ceiling priority" mean, since we can now change the priority dynamically? This needs to be answered in the proposal.

If tasks have a priority anywhere in the EDF range, you run at the bottom priority.

EDF will be optional, similar to the way that Round Robin is optional.

Add **procedure** to the procedures in package Deadline_Support; make Get_Deadline a function.

Alan doesn't think that it should be a child of "Dispatching" because it is useful for other purposes. But those purposes would still be scheduling issues. So make it a child of "Dispatching". And the implementation advice seems unnecessary and potentially expensive. Call the package "Deadlines".

The deadline stuff is really separable. Any task could have a deadline; it's not specific to EDF.

Keep the AI alive: 9-0-4.

Alan will create a new version of the AI.

AI-361/01 Raise with message

Pascal explains the AI. He says that the note represents a bad idea, and should be removed.

Ed notes that the wording of 11.3(4) is strange. Ed suggests: "If a string expression is present, a call to Ada.Exceptions.Exception_Message returns that string."

Erhard wonders about **raise**; this paragraph doesn't say anything about **raise**, with respect to Exception_Message. No, that's covered by using the existing exception occurrence.

Jean-Pierre notes that **raise** E; is different from Raise_Exception (E'Identity); Raise_Exception (E'Identity) has an empty Exception_Message, while **raise** E; has an implementation-defined Exception_Message. Yes, that has always been true.

Approve AI with changes: 9-0-2.

AI-362/01 Some predefined packages should be preelaborated

Randy explains the reasons behind his reasoning. It is all laid out in the discussion.

The group is in favor of more categorization for packages. But there is concern that some of the changes may cause real problems for implementers.

Action item: Implementers should give feedback on whether any of the proposed changes and including Calendar/Real_Time is a problem for the implementation.

During the discussion it is also noted that access-to-subprogram types and access types with no storage pool should really be allowed in pure units. Erhard will create an AI on this topic.

Keep it alive: 13-0-0.

Randy will poll the implementers and then update the AI with the results.

AI-363/01 Eliminating access subtype problems

Tucker explains the problems (then leaves for the airport).

Pascal thinks that these issues deserved to be fixed. He points out that the privacy breaking case has happened to him several times.

We decide to defer further discussion as Tucker has left (and no one else understands the problems well enough to lead the discussion). Tucker will update the AI if needed.

AI-364/00 Fixed point multiple/divide

The problem is that you can declare user defined "*" and "/" for fixed point types, but you can't call them. Tucker has customers that have stuck with Ada 83 specifically because of this problem with Ada 95. A previous AI confirmed the language here, mainly because no easy solution was apparent. Tucker now has a possible solution.

Steve asks if there is a visible fixed point type and an operator declared in the private part. That will get different answers in different places. That's probably OK, but the wording has to be careful.

Approve intent of AI: 10-0-2.

Tucker will write up the AI.

AI-365/01 Permissions to create grandchildren of Ada

Typo "decendant" should be "descendant" in the problem.

Randy describes his position — user-defined code shouldn't be mixed up with language-defined code unless absolutely necessary.

Pascal would like to have a statement reserving these hierarchies for the standard and the implementers: users should keep their hands off.

Greg says that the user may want to compile the body of Text_IO. That's not a problem; it can be done in a non-standard mode.

Pascal would prefer to change "child" to "descendant" in A.2(4). Ed thinks that is it OK to leave it up to implementers whether to allow great-grandchildren of Ada.

Pascal doesn't understand A.2(4), he claims it is inconsistent: why are only children of Ada covered? Ada.Numerics can't be recompiled, but Ada.Numerics.Generic_Elementary_Functions can. Ed thinks that there is value to allowing replacement of implementation.

Erhard says that the rule catches the very naïve user who thinks that every Ada reusable package has to start with Ada. Pascal exclaims that this is the weakest argument that he has heard in a long time.

Pascal thinks that either A.2(4) should say “descendant”, or it should be deleted and we should just rely on A(4). Several people wonder about the value of making a change here.

We take a straw vote: Leave A.2(4) alone: 7; Change A.2(4): 3; Abstain: 1.

No action: 9-1-1 (Pascal still objects for the reasons detailed above).

Detailed Review of Regular AIs

AI-188/03 The definition of setting a task base priority is too vague

Alan would prefer this to be a “no action”. Ted Baker argues in the !appendix that the vagueness is entirely intentional.

No, this isn’t a “no action”; the question is entirely reasonable and deserves an answer, even if the answer is “the wording in the standard is correct”.

Alan will rewrite the AI as a confirmation AI.

AI-279/03 Tag read by T’Class’Input

Typo in the question: “the the call”.

On 3.9(12) - we don’t elaborate freezing points. Perhaps we could say “freezing point has not been reached”. Or just “frozen”? No, freezing is a static notion, and this is dynamic.

Ed wonders if fixing this is really important. Pascal says that this is serious safety problem, it will happen very rarely, but it when it does, things are very bad.

Approve intent of AI: 7-0-4.

After the meeting adjourned, a number of ARG members continued to discuss this AI (it was the last one discussed). Pascal suggested “execution has not reached the first freezing point” to fix the wording.

Ed suggests adding something to the freezing point wording (13.14) to say that the tag of the type is elaborated when the type is frozen. Then it is OK to say that Tag_Error is raised if the tag is not elaborated.

Erhard suggests adding a bounded error if you call Internal_Tag before the main progra m starts execution. Users will not be happy if Tag_Error is raised occasionally, it would be better that it always be raised. The bounded error would say that it works or raises Tag_Error (or of course, raises Program_Error). This is easier to implement as well.