Minutes of the 23rd ARG Meeting

13, 14, 17 June 2004

Palma de Mallorca, Spain

Attendees: Steve Baird (except Sunday morning), John Barnes, Randy Brukardt (except Sunday morning, most of Sunday afternoon), Alan Burns (except part of Thursday afternoon), Kiyoshi Ishihata, Pascal Leroy, Erhard Ploedereder (except short parts of each day), Jean-Pierre Rosen (except Monday); Ed Schonberg (Thursday only), Tucker Taft, Joyce Tokar (except Monday afternoon and second part of Thursday morning), and Tullio Vardanega (except part of Thursday afternoon).

Observers: Javier Miranda, Matthew Heaney (Monday afternoon only)

Meeting Summary

The meeting convened on 13 June 2004 at 09:00 hours and adjourned at 19:25 hours on 17 June 2004. The ARG did not meet on 15 June or 16 June in order to allow attendees to attend the Ada Europe conference. The meeting was held in meeting room Zeus at the Melia Palas Atenea hotel, in Palma de Mallorca, Spain. The meeting was hosted by the Ada Europe conference. The meeting covered most of the agenda, discussing all amendment AIs on the agenda, as well as the majority of normal AIs.

AI Summary

The following AIs were approved:

AI-100/05 Truncation required for expressions if Machine_Rounds is false (11-0-0)
AI-294/04 Instantiating with abstract operations (8-0-1)
AI-297/09 Timing Events (8-0-0)
AI-341/01 Primitive subprograms are frozen with a tagged type (8-0-1)
AI-349/01 Equality on private extensions (9-0-0)
AI-376/01 Interfaces.C works for C++ as well (3-0-3)
AI-378/01 The bounds of Ada.Exceptions.Exception_Name (6-0-0)

The following AIs were approved with editorial changes:

AI-188/05 The definition of setting a task base priority is too vague (10-0-0)
AI-204/04 Language interfacing support is optional (10-0-0)
AI-214/02 Distinct Names for compilation units (again) (9-0-0)
AI-239/02 Controlling inherited default expressions (8-0-1)
AI-266-02/06 Task termination procedure (6-1-2)
AI-280/03 Allocation, deallocation, and use of objects after finalization (8-0-1)
AI-286/07 Assert pragma (9-0-0)
AI-287/06 Limited aggregates allowed (9-0-2)
AI-307/08 Execution-time clocks (8-0-0)
AI-317/08 Partial parameter lists for formal packages (11-0-0)
AI-318-02/02 Limited and anonymous access return types (9-0-2)
AI-320/02 Violating Ada semantics with an interfacing pragma (8-0-0)
AI-327/05 Dynamic ceiling priorities (8-0-0)
AI-335/01 Stream attributes may be dispatching subprograms (6-0-0)
AI-344/03 Allow nested type extensions (8-0-0)
AI-345/05 Protected and task interfaces (11-0-1)
AI-354/04 Group Execution-Time Budgets (7-0-1)
AI-359-02/01 Deferring Freezing of Generic Instantiation (6-1-4)
AI-362/03 Some predefined packages should be recategorized (9-0-0)
AI-363/03 Eliminating access subtypes problems (8-0-1)
AI-364/03 Fixed-point multiply/divide (7-1-1)

AI-368/02 Restrictions for obsolescent features (9-0-0)
AI-381/02 New Restrictions identifier No_Dependence (11-0-1)

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

AI-51/11 Size and alignment clauses (9-0-0)
AI-133/03 Controlling bit ordering (7-0-3)
AI-158/02 Renamings of primitives of a class-wide generic actual type (11-0-0)
AI-302-03/04 Container library (9-0-0)
AI-333/01 FIFO_within_Priorities does not require Ceiling_Priority (10-0-0)
AI-334/01 Is overriding of abstract equality required? (9-0-0)
AI-355/03 Priority Specific Dispatching including Round Robin (8-0-0)
AI-357/04 Support for Deadlines and Earliest Deadline First scheduling (7-0-2)
AI-366/03 More liberal rule for Pure units (6-0-6)
AI-370/02 Environment variables (6-0-3)

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

AI-162/04 Anonymous access types and task termination/controlled type finalization
AI-291/02 By-reference types and the recommended level of support
AI-373/02 Undefined discriminants caused by loose order of init requirements
AI-377/00 Naming of generic child packages

The following AIs were voted No Action:

AI-173/01 Optimizations and the use of 'Address (11-0-0)
AI-275/01 Aliased components and generic formal arrays (9-0-0)
AI-295/02 Another violation of constrained access subtypes (9-0-0)
AI-350/01 Allocating and comparing zero-size objects (5-3-1)
AI-356/01 Support for Preemption Level Locking Policy (9-0-0)
AI-359-01/03 Deferring Freezing of Generic Instantiation (9-0-1)
AI-380/00 Sign of zero in complex values (6-0-0)

Detailed Minutes

Meeting Minutes

The minutes of the 22nd ARG meeting were approved by acclamation.

Next Meeting

Randy will host the next meeting (24th) in Madison, Wisconsin, September 17-19. When Randy arrived, he asked if there was a preference between a downtown or suburban location, and described the choices a bit. A clear preference for a downtown location was shown by the group.

The following meeting (25th) will be held following the WG9 meeting at SigAda '04, in Atlanta, Georgia. The dates are November 19-21.

Discussion turns to the February meeting (26th). This will be an important meeting, as we'll be finalizing the Amendment if we stay on schedule. Tucker volunteers to host in Boston. Jean-Pierre renews his offer to host in Paris. Since the previous two meetings will be held in the US, a European location seems preferable. And Paris is lovely in February. :-) Dates of February 12-14 are tentatively chosen.

The summer meeting will again be held in conjunction with Ada Europe. The conference takes place the week of 20-24 June. However, if we stay on schedule there should be little technical work remaining to be done at that time, so it's unclear if a 3-day meeting is warranted. Pascal suggests that people reserve both weekends before and after the conference, and that we decide on a date later, once we have a better visibility of the progress made.

Old Action Items

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

Steve Baird:

- AI-51
- AI-109 [merged into AI-51]
- AI-373

Randy Brukardt:

- AI-302-3
- AI-303
- AI-309
- AI-362
- AI-368

Editorial changes only:

- AI-148
- AI-209
- AI-245
- AI-251
- AI-286
- AI-329
- AI-351
- AI-360

Pascal Leroy:

- AI-100
- AI-186
- AI-226
- AI-239
- AI-294
- AI-335
- AI-359 (possibility 2)
- AI-367 (to explain why it is No Action)
- AI-370

Erhard Ploedereder:

- AI-366

Tucker Taft:

- AI-86 [merged into AI-317]
- AI-318
- AI-344
- AI-345

- AI-359 (possibility 1.5)
- AI-363
- AI-364
- Review AI-351 for any violations of the Tucker model of time.
- Create an AI on the restriction No_Dependence (see the discussion of AI-353 in San Diego) [now AI-381].

New Action Items

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

Steve Baird:

- AI-51
- AI-291
- AI-373
- AI-377

Randy Brukardt:

- AI-279 (update to reflect the adoption of AI-344)
- AI-302-3
- AI-333
- AI-334

Editorial changes only:

- AI-188
- AI-204
- AI-214
- AI-239
- AI-266-2
- AI-275
- AI-280
- AI-286
- AI-287
- AI-295
- AI-307
- AI-317
- AI-318-2
- AI-320
- AI-327
- AI-335
- AI-344
- AI-345
- AI-354
- AI-362
- AI-363

- AI-364
- AI-368
- AI-381

Alan Burns

- AI-355
- AI-357

Gary Dismukes:

- AI-158
- AI-269

Pascal Leroy:

- AI-133
- AI-370

Erhard Ploedereder:

- AI-366

Jean-Pierre Rosen:

- Create an alternative to AI-284 where the keywords are reserved (and address support for pragma Interface). [Assigned at WG9 meeting.]

Ed Schonberg:

- Create an implementation report for AI-318.

Tucker Taft:

- AI-162
- Study and create an AI if necessary on the non-symmetrical nature of 4.6(16) (see discussion of AI-363).

Items on hold:

- AI-318-1 (This AI should be voted No Action now that it's alternative has been approved)

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-266-02/06 Task termination procedure

Alan says that he only half turned the procedure into a function; **out** should become **return**.

Pascal thinks that the subprogram identifiers are still not intuitive.

Alan notes that the third paragraph of the Dynamic Semantics wording is the important one.

Tucker suggests changing “Own_” to “Current_Task_”.

Change “Default_Handler” to “Fallback_Handler”.

Alan wonders if this needs a Minimum_Ceiling_Priority to ensure handler is callable. No, but if a task runs at interrupt priority level, the user must use an interrupt priority ceiling handler.

Jean-Pierre notes that the wording says “if the task terminated due to completing that last statement”; but we’re talking about a task that hasn’t terminated yet. So the wording should talk about completion, not termination. “If the task completed due...”. Similarly, it should say “if *completion* is due to abort...”.

Pascal asks what happens if a handler propagates an exception? The exception is ignored. Hence, this isn’t a foolproof mechanism. But, if you prove exception freedom in the handler (using SPARK, for instance), you can be certain everything is handled. Pascal grumbles that if you prove exception freedom, you don’t need this mechanism in the first place.

Alan asks if this can ensure that all tasks are handled. Deferring elaboration and doing it as part of package elaboration allows a fallback handler to be defined for the environment task, which effectively defines one for all tasks.

Jean-Pierre asks about the second-to-last paragraph of the Dynamic Semantics wording: “For all operations and types ...” — why “and types”? Types don’t raise exceptions. Drop “and types”.

Pascal: Why raise Program_Error on null task ID? Alan explains that it is consistent with task_attributes and dynamic priorities.

Approve AI with changes: 6-1-2.

Pascal voted against because he doesn’t understand how to use it, or why you need it. And he thinks that the model is awkward.

AI-286/07 Assert pragma

France submitted a comment on this one, and it seemed substantive, so the AI was reopened.

Jean-Pierre explains that the current write-up has a default value (the empty string) for the Message parameter of procedure Ada.Assertions.Assert. This means that this procedure will never provide the implementation-defined information that is normally provided by a raise_statement (stack trace, line number, etc.). This is unfortunate because this information is useful for diagnosing problems without a debugger.

So he proposes to have two overloaded procedures Assert: one with one parameter, which corresponds to a vanilla raise_statement, and provide in the exception message some implementation-defined information; and the other with two parameters, corresponding to a raise-with-message.

Tucker and/or Randy will produce wording prior to editorial review. Note that we now need two equivalence rules to describe the effect of the two procedures Assert.

Jean-Pierre says that the pragma has no name for the first parameter. He would prefer that the parameter had the same name as procedure Assert so that a programmer can just switch back and forth between the pragma and the procedure.

We take a straw poll on adding a name for the first parameter: 2-0-7. Since there is no opposition, we now have an argument name.

Alan asks that a real example be added.

Pascal asks that the reference to AI-324 in the discussion paragraph labeled “1)” be removed, since that AI is essentially dead.

Approve AI with changes: 9-0-0.

AI-287/05 Limited Aggregates Allowed

IBM/Rational had a substantive comment on this AI, so it was reopened.

Pascal explains that the wording is problematic where it uses the term “assignment operation”. Assignment operation is a dynamic semantics concept, it is not good for legality rules. Moreover, it introduces a big incompatibility because by-copy parameter passing involves an assignment operation.

Pascal recommends that the wording enumerate individual cases. Indeed, the AI already enumerates them in the AARM note, except for aggregates and return statements, where we need to decide.

We decide to defer any rule on return expression to AI-318-2. Tucker suggests that we adopt wording similar to that used in AI-318-2.

Tuck and Pascal will develop wording in time to review on Thursday.

Pascal suggests gathering all rules about limited in 7.5. They are currently scattered throughout the manual, and most of them need revision. It is better to group them in one single place, and be more explicit about the individual cases.

John comments that there are a number of typos, which he will send to Pascal/Randy during editorial review or sooner.

AI-287/06 Limited aggregates allowed

Tucker explains the wording changes. The change to 5.2 is not absolutely necessary, but it makes sense for these to be all legality rules, rather than name resolution rules.

This is not quite compatible: it’s possible that existing code would become ambiguous. Such code is pretty pathological anyway, it would have to have overloaded limited and non-limited functions, with the non-limited one being selected.

We discuss AI-318-2 to understand the bigger picture.

Erhard is concerned about the wording of the new 5.2(4): “expected to be of any type”. That may be ugly, but occurs elsewhere, such as 4.6(4). Erhard withdraws his comment.

Approve AI with changes: 9-0-2.

This AI will go back through editorial review again.

AI-297/09 Timing Events

Alan explains that the definition of time was enhanced with advice and consent of Randy

The third paragraph of dynamic semantics says “as soon as possible” — what does this mean? In metrics, a measurement is required to give some precision to this.

Does anyone provide any metrics? Joyce says DDC-I did provide them. GNAT and Rational do not provide them, as there are too many combinations. They’ll provide them on special request.

A vote is postponed until Randy arrives.

On Monday, we take the vote.

Approve AI: 8-0-0.

AI-302-03/04 Container library

How should be Size and Resize be changed? Pascal would like the same term for both. “Capacity”, “Set_Capacity” is decided on. The type should be named “Count_Type”, as it is used for Count parameters.

Set_Capacity can shrink the vector or other container; there is no good reason to preclude that. The value returned by Capacity is greater than or equal to the value passed to the last successful call to Set_Capacity.

Tucker wonders why some of these operations have **in** parameters for the container. Matt says the model is that changing an element does not change the container itself.

The name Generic_Update is uncomfortable. We already decided on the name Update_Element in Phoenix; but Matt changed it because it didn’t have the prefix “Generic_”. Do we need the prefix “Generic_”? Yes (it’s consistently used in the containers, and in most other generics in Ada as well). But we still need “Element”. Call the routine “Generic_Update_Element”.

Do we need Generic_Delete, Generic_Find, etc.? Matt tries to explain the need for the predicate. Randy suggests that KISS (Keep It Simple Stupic) applies here. Pascal agrees. These are non-primitive operations (they can be easily be written based on other operations in the packages), so drop them. Also drop Delete by item.

Matt explains the idea behind Generic_Query. There are other ways to do this (it’s also non-primitive); we don’t need it.

Do we need Reverse_Vector to be similar to Reverse_List? Reverse_Vector seems useless and expensive, while Reverse_List seems useful, so leave this just as is.

Swap for list: After a swap, do the cursors still designate the same elements, or do they designate the swapped elements? The semantics should be similar to that for an array: much like an index designate a “box” in the array, a cursor designates a “box” holding an element, so after a swap, the elements in the boxes should be changed. So we swap the elements, not the nodes.

Pascal asks why there are <> defaults on generic formal subprograms? He prefers that we use defaults only on operators, named operations usually have names designed for use as a formal (as opposed to as actual routines). Others note that the examples are confusing because of their heavy use of the <> notation. We agree to drop the <> from named routines.

Map equality is weird. It ignores the keys. Equality is comparing a range/domain to another range/domain — so both the range and domain ought to be the same. So the keys must be in both, and they must designate the same element. That definition is expensive — it’s O(N), but that isn’t important — equality is mainly defined because Map is a non-limited type, and we have to define equality for such a type. Tucker would like to use predefined equality for Keys.

Matt asks whether there should be a box for Hash. Is_Equal_Key? That really is equivalence of keys (rather than strict equality). Add a formal "=" for Key_Type before Is_Equal_Key. Change Is_Equal_Key to Equivalent_Key. Leave the default for Equivalent_Key "=" (which is the formal above it).

Looking at the Set package, we turn to the Lower_Bound and Upper_Bound operations. We deleted these in Phoenix, but discussion on Ada-Comment indicates that the functionality is needed. The names Lower_Bound and Upper_Bound are confusing, especially because they’re not inverses.

Floor and Ceiling would be OK. Floor is <=; Ceiling >=.

Tucker would like Equivalent to be passed into Sets the same as for Hash. Matt thinks that’s confusing, "=" is *not* the same as Equivalent; Equivalent is (**not** (A < B)) **and** (**not** (B < A)). The text is supposed to say that, but doesn’t. It does say that in Insert; fix the wording there to not try to redefine it, and define it properly at the top.

Change the library level requirements to say “will be dropped” instead of “can be dropped” (so no one thinks that it will be permanent.)

Should we have a hash function test capability? Matt explains why it would be useful. But debugging isn’t that hard, so we don’t want to add it here.

Pascal asks about Generic_Update for a set changing the ordering function. That is erroneous; checking is too expensive (certainly not O(1)). Pascal objects, because it damages the safety of the abstraction. Tucker suggests that renaming it as Generic_Unchecked_Update to make it clear that it is unsafe.

Pascal suggests that the routine check that it is equivalent to the previous version. That would work, but you'd have to copy the element to do it, and that would defeat the purpose of the routine (to avoid copying the element)

We need to achieve safety in this container. Drop Generic_Update if necessary to achieve this. But erroneous behavior is not acceptable. Tucker volunteers to try to find an alternative to dropping it.

Do we need Generic_Insertion? It seems to be about the same as Insert, since it cannot trust Set_Key. Matt says it inserts first; no, that would be erroneous if Set_Key is bad. And we certainly don't want erroneous behavior. Delete Generic_Insertion.

Pascal asks if routines like Generic_Update_Element should use an anonymous-access-to-subprogram (AI-254) instead of a generic. This would look like:

```
procedure Update_Element
 (Position : in Cursor;
   Process : not null access procedure (Element : in out Element_Type));
```

We couldn't do that with Sort or anything else that takes operators (they're intrinsic and thus don't match). But we could do it with Generic_Iteration.

Does the access procedure go first or last? Tucker says the Process parameter goes first. Pascal agrees. But Matt, John, and Randy disagreed.

Tucker points out one reason not to use AI-254 is that this can be implemented in Ada 95 tomorrow (such as Matt's implementation). But that shouldn't be used as an excuse to make it harder to use the standard.

We take some straw polls on this idea.

Generic: 2; anonymous access: 4; abstain: 3.

Process (the procedure) goes first: 2; process goes last: 3; abstain: 4.

So, go to anonymous access rather than generics, the process parameter goes last.

Why this not tagged? The implementation will be tagged. Give the user the maximum capability. Moreover, with the container first, the users could use Object.Operation notation for calls, which makes sense for these operations. Matt would prefer these aren't tagged, but he doesn't seem to be able to give any reason for that view.

We take a straw poll: Containers should be tagged private: In favor: 4; Opposed: 0; Abstain: 5.

Approve intent of the AI: 9-0-0.

On Thursday, Tucker explains his proposed solution to Update_Element problem with Set.

He suggests providing functions to Generic_Keys (Key_Of, "<" for keys), and then Update_Element would raise an exception if the Key is changed to be non-equivalent. Steve points out that you can't leave the element there when it raises an exception, it would have to be deleted. That's ugly, but it can't be helped.

The comparison would be equivalence (**not** (A < B)) **and** (**not** (B < A)), not equality. (No sense in making a stricter requirement than needed.)

AI-307/08 Execution-time clocks

Alan explains that the package was split into two parts:

- parent package Execution_Time — measurement of computation time

- child package Timers — can cause a handler to be executed

John says that the summary is no longer appropriate: obsolete stuff should be in the discussion. Perhaps in a "historical context" or "rejected alternatives" section.

Pascal points out that there should be an explanation in the problem section about doing simple CPU time measurement, even in non-real-time programs. That will give a hint as to why packages are now structured this way.

Pascal wonders if there is a precendent for an "implementation-defined-real-number". Yes, see Real_Time in Annex D

Alan notes that **not null** is inserted once; there might be other places where it goes.

Tucker suggests that this shouldn't be done piecemeal; it should be done systematically as part of the integration task. That is, the insertion of **not null**, **access constant**, etc. should be checked in the reference manual as a whole, including all AIs.

Approve AI with changes: 8-0-0.

AI-317/07 Partial Parameter Lists for Formal Packages

Tucker explains that he incorporated AI-86, which dealt with the actual itself being a formal package. This is the final paragraph of wording in this AI.

Tucker and John note that the discussion needs words relating to partial "<>" situation. It is written presuming all or nothing. Same goes for example.

Pascal suggests giving names to entities in the new paragraph to help the reader. Tucker will produce new wording by Thursday.

John notes that AI-324 is referenced in the example; keep the reference but identify the AI as now defunct.

AI-317/08 Partial parameter lists for formal packages

On Thursday, Tucker explains his updated wording for the second paragraph of 12.7(10), where he now gives names to the entities mentioned in the paragraph.

Approve AI with changes: 11-0-0.

AI-318-02/02 Limited and anonymous access return types

Anonymous access types now have accessibility of the function; nothing dynamic. This is simpler, and works like anonymous components.

Implicit dereference is allowed for function calls. This is OK because there is no assignment to the pointer (you couldn't do it for variables, because then you couldn't assign to the variable).

Finally, we add extended return (as previously discussed).

Steve notes that the rule that makes unconstrained returns illegal is incompatible; you couldn't return an unconstrained array of tasks. Tucker points out that if those are returning existing objects, they'd have to change to returning anonymous access to unconstrained anyway. So there is no additional incompatibility here. Steve says that you couldn't return a slice of a global array of tasks this way. True, but that seems more like a pathology than an important capability.

Do we want the restriction on limited return to be "constrained" or "definite"? We want "constrained", because objects of limited types are typically allocated to size, as they can't change afterwards. If we used "definite", we wouldn't know the size until after the function reached its return statement.

Pascal asks if the legality rule is privacy breaking. Tucker says it is not, because the caller and the callee would both see the same constraint. If the full view of a constrained partial view is unconstrained, it is at least definite. The implementation would have would use allocate-the-max for such a type (or use non-contiguous allocation).

Tucker says that the caller does most of the work of allocating and setting up memory.

Pascal asks how the constrained rule is handled for a generic unit. How is that checked for a formal limited private type? A function returning an unconstrained array of formal limited private would be illegal. Pascal says that this is a big incompatibility. Tucker says that doing that would be a violation of the contract (if the function returns local objects) — the function would raise Program_Error if the generic was instantiated with a limited type. But that "violation" would only show up at runtime, so people could depend on a generic containing such a violation.

Pascal notes that we are creating this incompatibility just to avoid implementation complexity — that doesn't feel comfortable. The root of the problem is that we're disallowing returning limited local objects. Tucker says that has nothing to do with the constrained limitation.

Note: After the meeting, Pascal points out that this limitation would make ACATS foundation FDD2A00 illegal; this unit does not raise Program_Error in Ada 95. This example shows a conflict between this limitation and the stream attributes, which will need to be resolved.

We take a straw poll — can live with new capability having the constrained rule: 6; cannot live: 2; abstain (zombies? There doesn't seem to be a middle ground here): 4.

Steve asks what happens if the subtype has a per-object constraint. He writes an example after prodding.

```
type T (D : Integer) is limited
    record
        F : String (D .. 10);
    end record;

subtype S is T (-1);

return O : S do ...
```

The result object should have already made this check. If it was returning into a temporary (as in parameter passing), it doesn't matter whether the exception was inside the function or outside, as you couldn't tell.

Pascal comments that implicit dereference would be irregular; why are functions special? Erhard wonders if the implicit dereference is the same for all access types? No, it only works for anonymous access functions — otherwise there would be a lot of ambiguity.

Steve wonders if there is a case where this rule would make something ambiguous that would otherwise be legal. A = B is suggested (both operands are functions with anonymous access type).

Tucker says the feature is kind of cool, and writes an example that supposedly demonstrates the "coolness":

```
function Str (Arg : ...) return access String;

Put_Line (Str (X)); -- With implicit dereference.
Put_Line (Str (X).all); -- Without implicit dereference.
```

Since we're converting:

```
function Lim (Arg : ...) return File_Type; -- Existing
```

into:

```
function Lim (Arg : ...) return access File_Type; -- Ada 2005
```

Pascal notes that if there is another

```
function Str (P : ...) return String;
```

Then the call

```
Put_Line (Str (X)); -- Is ambiguous if implicit dereference

if Lim(A) = Lim(B) then -- Would be ambiguous.
```

Jean-Pierre comments that from a training perspective, it would be very hard to explain the difference between named and anonymous access types in function results.

We take a straw vote. Writing .**all** is OK: 9; Have implicit dereferencing for functions returning anonymous access: 2.

Remove everything that talks about changing 4.1 to implement implicit dereferencing.

Approve AI with changes: 9-0-2.

AI-327/05 Dynamic ceiling priorities

Alan describes the changes. There is a general statement in D 3 that it is a bounded error for a task to be queued on any protected object with a lower ceiling than the task's active priority. This replaces D.5(11), which is removed. New section D.5.1 is a tighter definition, metrics are now provided.

John notes that "when entry opens" isn't good wording; use "when the entry is open". Tucker suggests "once the entry is open".

Pascal would prefer to see these sections reorganized. Make D.5 Dynamic priorities, D.5.1 for tasks, D.5.2 for protected objects. Erhard wonders if we should mix static and dynamic priorities. No, two separate subclauses should be used, D.5 is just an introductory sentence.

Erhard asks about the restriction identifier, it doesn't even allow querying the priority. That's intentional, because 'Priority is a variable. There's no precedent for a restriction changing a variable into a constant. In addition, this is consistent with the rule for tasks, where Get_Priority is disallowed as well. Erhard is satisfied with these answers.

Pascal wonders, since the "attribute denotes a component", is the component **aliased**? No. John asks whether the word is "non-aliased" or "nonaliased" — we agree to let him decide.

Summary: Restructure D.5 to D.5.1 for tasks, D.5.2 for protected objects. Add "non-aliased" to component definition.

Approve AI with changes: 8-0-0.

AI-344/03 Allow nested type extensions

Someone asks to have the e-mail discussion summarized. The e-mail discussion was about tags. For streaming, the current rules are good enough (the problems that could arise are no worse than streaming in junk, which always can happen). But for dispatching, we have to be able to identify a specific elaboration of a type.

So a tag cannot be completely static. But we don't want to expose this mess to the user by having two different kinds of tags (as was suggested at one time). Nobody cares if the tag exported by Ada.Tags gets bigger.

Pascal wonders if Bob's suggestion that a tag identify a type rather than a type_declaration is a good one. That would require changing the representation of existing tagged types, and that does not seem like a good thing here. Pascal thinks that 3.9.4 is messy as is, and probably will require changes to existing types. Tucker says that if you don't use nested extensions, you don't want the program to slow down because of them.

Allocators have to make a run-time accessibility check if you have an initialized class-wide object. That's the rule change in 4.8(7).

Randy notes that the changes to 'Input will require a redesign of 'Tag_Read and 'Tag_Write. He's told that's integration — it won't be handled separately (he's thrilled).

Tucker comments that the restriction on T'Class'Input not returning objects of a shorter lifetime (implied by the definition of Descendent_Tag) is similar to the restriction on other functions (in 6.5(20)). That is a good point, it should be mentioned in the discussion. [Editor's note: It's not clear how these restrictions are similar beyond the similarity already noted in the AI's discussion.]

Steve reports that all of the concerns that he had have been addressed.

There needs to be an example.

Pascal thinks that there are cases where we allow restriction to library-level of generics. Randy says that there is an AI, AI-115, that says (see A.5.2(27.d.1/1)) that all language-defined units must allow nested instantiations. The AARM notes for AI-115 will need to be updated.

Approve AI with changes: 8-0-0.

AI-345/05 Protected and task interfaces

The AI added synchronized interfaces, and updated the syntax to make task and protected interfaces clearer.

Jean-Pierre wonders why we don't have entries in interfaces. It doesn't work well with interfaces of various kinds — like a queue interface — with active, protected, and sequential versions. You want to be able to implement these the same way.

Erhard wonders about 9.1(8): "at most one" seems weird. The group agrees that it is correct. Better wording would be welcome, but nothing tried seems to work.

We turn to discussing the rule for allowing select statements on procedures that are not statically known to be a procedure. Tucker explains the rules again.

Jean-Pierre wonders whether access-to-subprogram should be included here. No, because using an entry wouldn't be possible; the convention would not match and the 'Access would be illegal.

Erhard wonders why the wording says "possibly one of the parameters". It should say "possibly the first parameter".

Add an AARM note to explain why it says "possibly". (For renames and some cases of formal procedures, the task object and entry are implicit in the name, otherwise the object is an explicit parameter).

Erhard wonders if "implemented by an entry" is confusing; renaming doesn't seem to be "implemented by". Tucker points out renaming an entry-as-a-procedure is an Ada 83 capability.

Pascal would prefer that the static semantics paragraph 9.7.2(3) be rewritten to explain each case explicitly, to avoid the use of "possibly". Also include more examples in the discussion.

Jean-Pierre asks what task and protected interfaces buy. Tasks can be aborted, the task id retrieved, etc. Protected specifies that there is no thread in the implementation. Synchronized says that the interface is always thread-safe. Thus these are more for user declaration than for additional properties, and these declarations are enforced by the language.

Pascal wants to ask about limited interfaces being implemented with non-limited types. Is there a problem? Steve does not remember any problems other than return-by-reference, which has now been deleted (by AI-318-2). It seems more universal to allow non-limited to match them.

Erhard wonders if **limited** is going to be used virtually all of the time. If so, it should be the default. Yes, probably, but making it the default would be inconsistent with the rest of the language. Generic private formals are the same way, but you have to say **limited**.

Alan points out that Tucker forgot "**is abstract**" on all of the operations in the first example.

Erhard would like "**is abstract**" to be implicit. Tucker disagrees; it clearly states that there is no body. Pascal notes that having it be implicit would make switching between abstract types and interfaces more painful.

The discussion should talk about the changes to syntax that Tucker made, and the change to allow non-limited to implement limited.

Add wording to say that **synchronized** is a new keyword.

Alan and Tullio think that this would useful to them as real-time programmers.

Would this be a problem to explain (to students)? Certainly it is similar to what Java provides; there is nothing new here. There is no inheritance, but adding that would be a real nightmare.

Approve AI with changes: 11-0-1.

AI-354/04 Group Execution-Time Budgets

Alan explains the changes. The type needs finalization, there were parameter name changes, not null was added where appropriate. The type names also were changed, and the minimum handler ceiling priority wording was changed. There were no major changes.

Tucker notes that "needs finalization" requires a systematic run-through some day. [Editor's Note: AI-360 which defined "needs finalization" did that for the Standard and all AIs existing at the time. The only AI added to the Amendment since is Containers, which has the necessary wording.]

John asks, in the example, why is parameter GB in Budget_Expired **in out** rather than just **in**? We agree it should be **in**. Tucker suggests renaming this routine to Budget_Has_Expired to make it clearer what it does.

Pascal wonders if Task_Array should be defined in Task_Identification? The group decides that it is insufficiently broken. But the declaration of Task_Array should be `Positive range <>`.

Approve AI with changes: 7-0-1.

Pascal is not sure this is useful. Alan mentions Philips needs this capability to isolate one group of tasks from another in a multimedia application.

AI-355/03 Priority Specific Dispatching including Round Robin

Tucker and John suggest reorganizing this to incorporate D.2.7 into D.2.2. Make policy-specific pragma an alternative to task-dispatching-policy, rather than using the "meta" policy Priority_Specific. Task_Dispatching_Policy is equivalent to using Policy_Specific to set one policy for the full range of priorities.

John notes that the pragma Pure should be for Dispatching, not Ada.

In the text of this AI, at many places, "Priority_Policy" should really be "priority policy".

Pascal wants to discuss the Priority_Specific_Dispatching package. He wants to know what are the benefits of being able to query the dispatching policy at execution time. Are there really circumstances where an algorithm will "adapt" to the policy in use? After some reflection Alan confesses that he cannot think of a possible use for this package, so the group agrees to get rid of it. Tucker is not fully convinced.

The post-compilation rule for round-robin (which requires ceiling locking) is useless because there is a similar rule for priority-specific dispatching. Remove it.

A few typos: in the 1st bullet of the dynamic semantics of round-robin, "when {a} task"; in the 5th bullet, "action" should be "active"; in the 3rd bullet "{See} D.14".

Pascal notes that "integer" should be "Integer" in name resolution rules for D.2.7.

In the second paragraph of the legality rules for D.2.7, “greater {than} or equal”.

Move dynamic semantics from D.2.7 into particular dispatching section when appropriate. The first three paragraphs stay with D.2.7 (which will be merged into D.2.2), the rest move into round robin or earliest-deadline-first (EDF), as appropriate.

An exception should be raised when attempting to set the quantum on a non-round-robin level. We need a query, Is_Round_Robin(Priority) in Round_Robin_Dispatching, to be able avoid raising the exception.

This is a general issue; we should add an exception Dispatching_Policy_Error into Ada.Dispatching for operation invoked on priority level or task for which policy is inappropriate. (Such as Set_Quantum for a non-round-robin level).

Pascal wonders where to put the note from the end of D.2.7. It should go into EDF and perhaps replicate in round-robin and FIFO subclauses if needed.

Pascal thinks that the example is lame. John disagrees, he feels it is helpful, Perhaps the example should specify FIFO for higher level priorities explicitly to make it clearer what is going on.

Approve intent of AI: 8-0-0.

Alan will update the AI. It is noted that this had better be updated soon and be ready for approval in September.

AI-356/01 Support for Preemption Level Locking Policy

Alan notes that this has been subsumed by the newer EDF approach.

No action: 9-0-0.

AI-357/04 Support for Deadlines and Earliest Deadline First scheduling

Alan notes that he gave a paper on this topic at Ada Europe and also a paper to Transactions on Embedded Systems. He handed out a paper copy.

John notes that **procedure** should be omitted on constant Default_Deadline.

Erhard notes that the pragma does set a deadline, but it is described as “enabling” in the proposal (in the introduction to the pragma). That seems confusing; it should be reworded.

Erhard finds it strange that there is no dynamic semantics associated with the pragma.

Tucker thinks that the pragma should be defined after the package. (Yes, that means having some syntax after the package).

Pascal wants the introductory paragraph beefed up. Have a separate normative definition of the policy identifier which is static semantics, and move the post-compilation section way down.

Is the policy identifier appropriate? The group would prefer to stick with EDF_Across_Priorities (Tucker and Pascal don’t love it).

Jean-Pierre notes a typo in rule 2: “deadline {of} the task”. Also, this rule should not say “T'Base” (that’s a type!). There needs to be real words like “base priority of task T”.

Alan asks that the second bullet in the Dynamic Semantics wording be reviewed. It is suggested to use “on ready queue with highest priority R” here, because R is a priority, not a ready queue. Or better, start from the wording in rule 2 of the proposal and use sub-bullets when necessary.

Pascal wonders why we need Delay_and_Set_Relative_Deadline. Alan says that it reduces unnecessary task switches by combining the two operations.

This is an absolute delay, not a relative delay. Tucker suggests changing the formal parameter name to "Delay_Until_Time" and change the name to Delay_Until_And_Set_Relative_Deadline. (Or a better name if Alan can come up with one). Also, capitalize the A in "and" in the name.

Someone points out that Set_Relative_Deadline has a race condition, so people could use it to shoot themselves in the foot. We better get rid of it, then.

Alan notes that the wording shouldn't refer to Any_Priority'First; it should always refer to the lowest priority of the EDF range. Pascal says that it is necessary to be careful with that. What happens if you have two separate EDF ranges, which happen to be adjacent? These are intended to be separate, and the wording should allow them to stay that way.

John asks that "above rule" be changed "preceding rule".

Jean-Pierre notes a typo in the paragraph before the discussion — "addition" should be "additional".

Pascal point out another typo in the 3rd paragraph of the discussion: "prioirity" → "priority".

John asks that "Q" be expanded to the word "queue". Similarly, Pascal asks that "pri" be expanded to "priority".

John notes a typo in the final paragraph of the discussion: "it did required" → "it did require".

Alan asks whether he should delete this part of the discussion. Pascal says that it is ok to keep so long as it is at the end, rather than in the middle of the main discussion.

John sees another typo: "Note {that in order} to be placed ...".

Pascal says that "PO" can be used in the discussion to represent "protected object".

Approve intent of AI: 7-0-2 (Steve abstains as he just arrived, and Kiyoshi abstains without comment).

Alan will update the AI.

AI-359-01/03 Deferring Freezing of Generic Instantiation
AI-359-02/01 Deferring Freezing of Generic Instantiation

Tucker explains alternative one. The pragma specifies that the instantiation is frozen at the point of the pragma. The unit is illegal if there is any freezing of items in the instantiation before the occurrence of the pragma. This preserves the current property of no access-before-elaboration checks.

The AI needs an example of using the pragma.

Tucker suggests possibly naming this pragma "Deferred_Elaboration" or "Deferred_Elaboration_Point", and placing it in chapter 12 (it's not really about freezing).

Pascal notes that the compiler would have to look forward for the pragma, preventing one pass compilation of instantiations. John notes that this is also a problem for the reader — there is no indication of the special semantics at the point of the instantiation.

Erhard wonders why the pragma isn't on the generic, rather than the point of the elaboration of body. That's essentially the other alternative.

Ed comments that it could be implicit. Others complain that the average user couldn't tell where the body elaborated in that case.

Turning to the second alternative. The only case where freezing matters is when a "two-part entity" is referenced, because the preelaboration rules cover the other cases. The rules essentially cover anything that could cause an elaboration bit to be checked.

Tucker suggests that the wording divorce itself from freezing, because it's really about elaboration bits. That sounds like a good idea. Essentially, everything is frozen (as it is now) when the body is elaborated.

Steve notes that preelaborate is a bit too strong, we could allow other things. But a new categorization pragma seems like too much for this capability.

Tucker says that both alternatives require doing piece-meal freezing of an instance (especially the specification). The first proposal requires extra checking of freezing. Pascal says that the first proposal would have a greater impact their compiler. Ed thinks that only one in a thousand user would understand how to use the pragma. He much prefers the second proposal.

No action on AI-359-01: 9-0-1.

The wording change would be: "freezing point" changing to "creation of an object of a type that does not have preelaborable initialization".

Tucker points out that you don't really need to defer the elaboration, because it doesn't have any real effect. Moreover, there can't be any access-before-elaboration problems.

Add an example to show how that fixes the problem. Check 3.11 and 3.11.1 to make sure that everything is covered vis-à-vis statically making elaboration checks, and that the wording works with renamings.

Approve AI with changes: 6-1-4.

Randy explains that he objects because splitting freezing from determining the representation of generics is very hard. Tucker thinks that there might be a solution to that problem — by adding another restriction, we might be able to defer the entire work to the body point.

After the meeting, this AI was returned to a work item because of e-mail discussions on the above and other points.

AI-362/03 Some predefined packages should be recategorized

The only change was to make System and System.Storage_Elements pure.

Jean-Pierre wonders if we need to keep this AI open, as the purity rules are still changing.

Pascal replies that there is no need, since preexisting visible access types cannot be changed at this point to have a zero storage size, and other AI-366 changes are already incorporated in the AI.

Tucker notes a typo in the first open issue, "persued" → "pursued". Pascal points out that this issue should simply be deleted (we're not going to pursue it).

Pascal thinks that the substitution rule for Wide_Maps must be changed, as it needs to indicate that it becomes an impure package. John notes that this is not a problem, since Wide_Maps has its own specification in A.4.7.

Jean-Pierre wonders if these packages are compilable. Pascal replies that Randy asked for review by implementors, and no complaints were lodged against these specific changes.

Approve AI with changes: 9-0-0.

[Editor's Note: The ARG approved this AI without addressing the other open issue, whether System.Address is being required to be transportable between partitions. Since requiring it to be transportable does not require additional wording, that choice is probably preferable; but the implications of that choice need to be included in the AI's discussion.]

AI-363/03 Eliminating access subtypes problems

Tucker explains the changes. He undid AI-168 (corrigendum AI), as it is no longer necessary. He also rewrote the unconstrained private type rule to be more compatible.

In 3.7.1(1) change “non-tagged” to “untagged”.

The reason that we need part 2 is that a partial view with no discriminants can have defaulted discriminants in the full view. Then a constrained object can match an unconstrained type. It’s too early to understand that, so Tucker writes an example.

```
type D (B: Boolean := False) is ...

type A1 is access all D(True);
type A2 is access all D;

P  : A1; -- A2(P) Legal? Yes, if all views of D are unconstrained.
P2 : A2; -- A1(P2) Legal? No.
```

Where is the partial view here? Tucker didn’t show it. If it is

```
type D (B: Boolean := False) is private;
```

then A2(P) is legal (because all views of D are unconstrained); otherwise, if it is

```
type D is private;
```

then A2(P) is illegal by the new rule.

Steve comments that this means that adding a **private** declaration could change the semantics of the full type, which seems weird. But that’s necessary for compatibility; we only want to change the semantics of cases with problems. It would be better to not constrain allocated objects, and deny these conversions and allocators always, but that’s way too incompatible to consider.

Pascal notes that there are other rules that change when a partial view is added (e.g., abstract operations in the private part).

Randy wonders if the new wording of 4.6(16) is missing a case (that is, no partial view at all). No, “any” partial view includes “no partial view”, so the existing case is included. Randy would like an AARM note to confirm this here (there is no change intended for the existing wording). Tucker wonders why that wording is there; conversions should go “both ways”, and any constraint checks should be done at runtime. No one knows, but we don’t want to introduce an incompatibility because of ignorance, so we won’t change or drop it.

4.8(6) allows unconstrained. John would like an example in the discussion of the AI.

Tucker notes with these changes, adding **aliased** to a declaration won’t change the semantics of the object, which seems like a good thing. This whole area is screwed up in Ada 95.

Straw poll on adopting part 2: 6-0-3.

Tucker will look at a “normal AI” to see why 4.6(16) does not allow things that would normally be checked at runtime — conversions should be symmetrical.

Approve AI with changes: 8-0-1.

AI-364/03 Fixed-point multiply/divide

Tucker explains the changes to the AI. The reference to user-defined “fixed-fixed” was eliminated. We now require “either” rather than “both” types having such an operator.

Pascal notes that Tucker didn’t implement John’s request to drop one of the words in “attempts to alleviate”.

Jean-Pierre asks why the proposal does not depend on the visibility of operators. That seems better. Tucker explains that depending on visibility causes Beaujolais effects.

Pascal asks that “univ fixed” in the discussion be changed to “universal_fixed”. John notes that “Beaujolais” should be capitalized. Use “operator” rather than “op”. John asks that “intended to provide” be changed to “provides”.

The brackets in the wording are confusing. Tucker explains that the first bracketed section had AARM brackets, and should be kept. Pascal asks that brackets not be used to mean deletion here.

Approve AI with changes: 7-1-1. (Pascal thinks there is too much implementation work given the benefit.)

AI-366/03 More liberal rule for Pure units

Pascal notes that there is a problem with distributed programs; we need to add a rule to E.2.3 to disallow use of “pure” access types in remote call interface packages.

An alternative solution is proposed: Add a requirement that all access types declared in a pure unit satisfy the requirements of E.2.2(8) (suitably modified).

Another option would be to require that Pure also be Remote_Types in order to use in a remote call interface package. But that would require going through the entire standard and deciding which Pure packages ought to be Remote_Types.

Tucker will take this issue off line, and will try to come up with alternatives which solve the problem with remote call interface:

1) no “hidden” access types in pure packages
2) require read/write attributes similar to that for remote types package
3) require remote types pragma on packages used for distribution
4) new local_pure categorization

On Thursday, Tucker explains his preferred alternative. He requires user-defined 'Read/'Write for any private type with an access type part in a Pure package. Then remote call type interface disallows parameters with access type parts that don’t have user-defined 'Read/'Write.

Steve wonders how this is checked for formal access types.

Randy wonders why this is forced in all pure packages. It seems a burden on users that don’t care about Annex E and/or streaming. Tucker claims that one of the properties of a Pure package is that they are always streamable. It’s a matter of safety; for limited, you can’t use streaming at all, whereas for non-limited, you could stream junk.

Erhard thinks that we should disallow all access type parts. He’s worried about dereferencing on another machine. Streaming operations should take care of this.

Approve intent of AI: 6-0-6.

Erhard will update this AI.

AI-368/02 Restrictions for obsolescent features

Pascal explains the changes — it is implementation-dependent if the restriction applies to J.1; the restriction is moved into chapter 13, it is not partition-wide.

Pascal notes a typo — “Many implementation{s} ...”.

Steve asks about generic instantiation. AI-286 talks about that for assertion policy, but it’s derived from the standard rule — the restrictions that apply to an instance are those that apply to the generic. In the case of this restriction there should be no additional checking on the instance, because the client has no control over the style used to write the generic.

Pascal notes that the words from AI-257 about not being partition-wide are missing here.

Erhard comments that all the examples where Restrictions itself is defined are not partition wide.

Erhard notes that the AI-257 wording will interact with this one. Perhaps it would be a good idea to factor out the wording about being not partition wide. That's an integration issue.

Approve AI with changes 9-0-0.

AI-370/02 Environment variables

Pascal simplified the proposal based on discussion; it was originally submitted by David Wheeler.

Tucker doesn't like the name "Ada.Environment", he prefers "Ada.Environment_Variables" or "Ada.Command_Line.Environment". Switch to "Ada.Environment_Variables" as the name.

Alan comments that this package should not be pure (as it has state), preelaborable is OK.

Pascal notes that there was an issue with positional access, since positions can change as a result of Set and Clear. It has been suggested to provide an iterator over names instead. A passive iterator (using a generic) would be the simplest.

Tuck notes that in Unix, an environment is an array of strings of the form "name=value", which allows duplicated names.

Randy had written that Count is expensive on Windows; there are two environments (system and user); and the environment is delivered as a giant string. He thinks that a generic iterator providing name and value would be best. (Both the name *and* the value must be passed to the formal subprogram of the generic, to properly deal with duplicates.)

Pascal notes that David Wheeler thought that a "clear all" operation is needed for security. That should be added.

Pascal points out that Clear by name removes the item as if it never existed. Tuck asks about duplicates. Pascal replies that you cannot create duplicates, but if they already exist, clear by name clears them all, "Set" first clears by name, and then adds a single new one. The wording should be modified to reflect this.

The group is uncomfortable with the fact that positions are invalidated by changes to the environment, so Pascal will remove all operations that take a Position.

Alan suggests that Clear all, clears the settable ones. That brings up the general issue of unsettable (that is, "system") environment variables Tucker suggests including a parameter in the generic iterator indicating whether the variable is settable, and a formal to select to only see the settable, or the non-settable ones. But there doesn't seem to be an easy way to tell if a variable is settable, other than trying to change it. So this idea doesn't work.

Tucker asks if the package allows concurrent queries on environment variables. Pascal says no, this is explicitly stated in the description of the package, if you want concurrent access you better implement protection yourself. Tucker wants concurrent reads to be supported, it's only concurrent modifications that are problematic.

Jean-Pierre wonders if we allow setting/getting even if there is no O/S support? Pascal replies, of course, if there is no O/S support, treat as though environment starts out empty, and if multiple processes are supported, should pass them on (in implementation advice), since that is the point.

Steve and Tucker both ask to remove the permission to modify the specification; it isn't needed here.

Approve intent of AI: 6-0-3. (People not voting cite a lack of knowledge about Windows, implementation advice may be inappropriate for Windows.)

AI-380/00 Sign of zero in complex values

Randy says that it would be hard to preserve the sign of the imaginary part (for example) if using a polar representation.

Pascal says that the Standard says to do something reasonable; trying to say more will take a lot of pages.

No Action: 6-0-0.

AI-381/01 New Restrictions identifier No_Dependence

Tucker explains that this AI is based on a comment from the 20th ARG meeting in Sydney related to AI-353.

An open issue is string literal vs. identifier name syntax. Tucker reports that AdaMagic will require new parser for the identifier case because it does overload resolution during parsing. A name that would denote nothing would be weird.

Should this apply to all units, to language-defined units only, or to language and implementation-defined units?

We take a series of straw polls:

String literal syntax: 1; name syntax: 4; don't care: 4.

Language and implementation-defined only: 3; any unit: 3; don't care: 3.

Forget AI: 5; Continue AI 3; Abstain:1.

Language-defined only: 4; Language and implementation-defined: 0; wish we'd stop talking about this AI: 5.

Pascal points out that if we kill this AI we will have at some point to invent a dozen new restrictions for the Alan packages, and that won't be more amusing. In particular, we'll have to discuss the name of each of these restrictions. So we better find a generic solution.

Keep AI alive, with language-defined only, name syntax: 7-0-2.

On Thursday, Tucker presents and explains his new wording. It's now a name (as agreed on Sunday), and language-defined only.

Steve wondered if it should say "needed" instead of "shall depend semantically". No, we want to encourage pointing out precisely where it goes wrong — a vague error message is not helpful.

Approve AI with changes: 11-0-1.

Detailed Review of Regular AIs

AI-51/11 Size and alignment clauses

In second bullet of the change for 13.3(30-32), say "chosen by default by the implementation", and get rid of the "i.e.".

Why is 13.3(31) changed so much? (It's now the last bullet of the changes for 13.3(30-32).) It should be as close as possible to the existing wording.

Pascal thinks that this should be a general statement in 13.1. That seems like opening another can of worms — users don't like this rule as it is — we don't want to rub it in their faces. Pascal withdraws his suggestion.

Requiring nonconfirming clauses for fixed point is OK.

13.3(56): “Similarly for” should read “Corresponding advice applies for…”. Fix summary and 13.3(30-32) as well.

Remove questions from the discussion section. (The second question seems adequately covered by the rules in 13.3; the other two we answered previously in this discussion.)

That leaves nothing much in the discussion. Steve says that there isn’t anything else to put here; the summary covers it pretty well. But it is pretty long for a summary. The summary should be simplified, and the details moved to the discussion section.

The summary should say that the rules currently require absurd things, and no implementations support those absurd things. The recommended level of support should match that. Users should have an indication of what they should expect (1.1.3(6) gives implementers license to ignore rules that are too hard in any case).

Approve intent of AI: 9-0-0.

AI-100/05 Truncation required for expressions if Machine_Rounds is false

Wording was added to define the machine numbers of a fixed point type (so that 4.9(38) makes sense for fixed point). Ed wonders if this will require implementation changes. No, this is only fixing a definitional problem, no language change is intended here.

Approve AI: 11-0-0.

AI-133/03 Controlling bit ordering

Pascal says that he took Norm’s paper and turned it into wording.

A “machine scalar” is really is register. Steve doesn’t like the term, because this is about the size of a scalar, not the item itself. Changing the term seems like a good idea.

Tucker says that Sofcheck requires that the position be on a word boundary; that seems to be a subset of the proposal. GNAT’s implementation seems to have incompatibilities, though.

Robert had said in mail that this solves an uninteresting problem. There is not much agreement with this position. John points out that his book says that this does not swap bits. Pascal suggests that solving the interchange problem would need a fancier solution, and would not use Bit-Order.

Normalization of 'First_Bit is bogus, because it could change the semantics of a later representation clause. We could say that you do not normalize in the non-default bit order. Randy agrees with this; the implementation should not lie to the user (giving some wrong answer).

Straw vote: is this problem worth solving? 7-0-3

What to do with storage place attributes? Should return the right thing, i.e. the value given in the representation clause in the case of the non-default bit order: 9-0-1.

Tucker suggests redefining “the first of the storage elements occupied” for non-default bit order to come from the specified position. Then no other wording needs change.

Approve intent of AI: 7-0-3.

Erhard is curious as to why the AI says “and no others”. Pascal notes that it wouldn’t make sense to define other things. Erhard says that if there is another kind of thing (say a string or complex) that is efficiently addressable, that is not allowed. It essentially says that there are some machine scalars that are not machine scalars. Fix this.

AI-158/02 Renamings of primitives of a class-wide generic actual type

This problem doesn't seem worth forcing implementers to do any work. Pathologies aren't tested, so we could give an answer and classify it as a pathology.

We prefer alternative 4 of the 4 presented. We'll classify the AI as a pathology, so no effort will need to be expended on this issue.

Approve intent of AI: 11-0-0.

Gary will update the AI.

AI-162/04 Anonymous access types and task termination/controlled type finalization

The problem doesn't look that important for this much wording change. But it is something that users can do, and they deserve to know what the semantics are.

Tucker will try to simplify this wording/description.

AI-173/01 Optimizations and the use of 'Address

Randy doesn't think that this is worth defining, because there is not much useful that can be said unless the objects are of the same type, and aliased. Implementers will do the right thing.

Tucker suggests that we only require anything special for objects which directly have address clauses; not for parameter passing. Otherwise we have distributed overhead.

There's not much interest in a feature that only works in very limited circumstances.

No Action: 11-0-0.

AI-188/05 The definition of setting a task base priority is too vague

Alan explains that Randy took care of organizing it in exchange for a pint. But he has to wait until York next year to get the pint.

Remove "the definition of" and "too" from the title. It's exactly as vague as it should be, and no vaguer.

Approve AI with changes: 10-0-0.

AI-204/04 Language interfacing support is optional

The parenthesized text in the first paragraph of the wording should be an AARM note.

The first paragraph should go after paragraph 10, as it's not Implementation Advice, but Implementation Requirements. The other paragraph should go after paragraph 11, Implementation Permissions. The existing paragraph should be just be deleted.

Approve AI with changes: 10-0-0.

AI-214/02 Distinct Names for compilation units (again)

Why define the term full expanded library name? The standard already uses "full expanded name" but doesn't define it. We don't need to do so here either, so drop that text. Just call it "full expanded name".

Change to singular: "…that contains a body_stub is added…" → "…as the body_stub."

Approve AI with changes: 9-0-0.

AI-239/02 Controlling inherited default expressions

Pascal wrote a new version recently. The only change is adding the words “a descendant of” into 3.9.2(18).

Last paragraph of discussion, last line, change “an concrete” to “a concrete”.

Approve AI with changes: 8-0-1.

AI-275/01 Aliased components and generic formal arrays

The problem that this AI covers is handled by AI-363.

No Action: 9-0-0.

AI-280/03 Allocation, deallocation, and use of objects after finalization

The AI was last discussed in Padua, updated by Randy soon afterwards.

There seems to be quite a bit of change in 7.6.1(11); Randy explains that this was simply to define *finalization of the collection* for later use. Less of a change would be preferred. Furthermore the new wording is broken as it would seem to imply that finalization takes place at the freezing point.

Replace the change in 7.6.1(11) by the original wording and then the following: “; the finalization of these objects is called the *finalization of the collection.*”

Remove the part after the semicolon in 4.8(11), Bounded Error, as was requested in Padua. The finalization of an object of such a collection has no effect, so no one cares when it is finalized.

Approve AI with changes: 8-0-1.

AI-291/02 By-reference types and the recommended level of support

(This AI was discussed immediately after AI-51, as the two are tightly related.)

Steve explains that there are many bizarre things that are required by the recommended level of support, and this fixes them.

Tucker wonders if “default value” is defined. No, the “i.e.” does so that. That’s no good. Default value isn’t used elsewhere, so get rid of it (and “aspect of representation”).

Put the most general paragraphs of 13.1(24) first.

Is the note normative text? This sounds like the definition of “could cause”, and it should be written that way. “For the purposes of these rules, *could cause* means …”. That means this is not a note.

Tucker notes that the 13.3(23) wording fixes the wrong thing; it should only change the definition of Alignment for subtypes (which is in 13.3(25)).

Note 5 (13.3(38)) in the existing Standard makes the claim that the alignment can be overridden. This needs to be normative, because there is no such rule. Use wording similar to that for Size in 13.3(48).

Tucker would like to split the description of Object'Alignment and Subtype'Alignment, as 13.3(23) is only about objects. 13.3(25) is mostly about subtypes. Given that Steve, Randy, and Pascal all misread the paragraph 23 as applying to subtypes, that seems like a good idea.

Is “unaliased” appropriate? Use whatever the standard currently does. By the wonders of modern technology, Randy and Tucker search the standard. Tucker finds “unaliased” in 13.3(52) — the only use. Other alternatives are checked; “non-aliased” is not used. We grudgingly agree that “unaliased” is OK.

A wording change to 13.5.1 comes before a 13.3 one in the AI; rearrange them.

This AI was not updated after the Phoenix meeting (it was accidentally left off of the action item list), so Steve should make sure that the comments from that meeting are included when he updates it.

AI-294/04 Instantiating with abstract operations

Pascal explains that after several unsuccessful attempts to change the normative wording, he determined that an AARM explanation was preferable.

Approve AI: 8-0-1.

AI-295/02 Another violation of constrained access subtypes

The problem that this AI covers is handled by AI-363.

No Action: 9-0-0.

AI-320/02 Violating Ada semantics with an interfacing pragma

Pascal does not like the way that this is phrased, because it implies that all Ada programs are erroneous.

Tucker wonders if we could just use the note text as a normative.

Pascal suggests something similar to 13.3(13). “If an Address is specified, it is the programmer’s responsibility to ensure that the address is valid; otherwise, program execution is erroneous.”

We decide to use the following after B.1(38):

Erroneous Execution

It is the programmer’s responsibility to ensure that the use of interfacing pragmas does not violate Ada semantics; otherwise, program execution is erroneous.

Approve AI with changes: 8-0-0.

AI-333/01 FIFO_within_Priorities does not require Ceiling_Priority

Alan explains that the rule is there to ensure that you have an analyzable system. But there is no reason to assume that some well-defined implementation-defined policy is not analyzable.

We list the possible resolutions, and take straw votes on “can live with this resolution” and “can’t live with this resolution” for each resolution.

1) No action: 2-1.

2) Fifo implies ceiling locking is default: 8-0.

3) Fifo means that some policy must be specified: 6-1.

4) Add an Implementation Requirement that the combination must be supported and delete D.2.2(5): 6-0.

5) Ceiling is the default: 1-3.

6) Delete D.2.2(5): 2-4.

Narrowing the choices to the three most popular, we take a preference poll. Prefer (2): 6; Prefer (3): 1; Prefer (4): 2.

The AI should be rewritten to reflect resolution (2).

Approve intent of AI: 10-0-0.

AI-334/01 Is overriding of abstract equality required?

Tucker says that this came up in a real program that was trying to hide "=".

The comment about “predefined "=" does the right thing” in the question contradicts the discussion; drop it.

The “notes on the wording change” references the wrong paragraph numbers — 3.9(4-6) should be 3.9.3(4-6).

Try to gather all of the wording changes in 3.9.3(4) and leave the bullets unchanged. Tucker will write this wording.

There are a number of typos. In the question, third line from the end: “for for” → “for”. In the wording notes: “Interesting this” → “interesting thing”. “A abstract” → “an abstract”. “predefine equality” → “predefined equality”.

Jean-Pierre wonders if the change would be better placed in 4.5.2. No, the text requires the “shall be overridden” wording, and that is in 3.9.3.

The discussion has carried on long enough that Tucker has already finished his homework. He reads the proposed wording. The group decides that it is too hard to decide on the correctness of this without being able to read it; besides, dinner is beginning to seem more important (this was the last AI discussed).

Approve intent of AI: 9-0-0.

AI-335/01 Stream attributes may be dispatching subprograms

Add “End AARM Note” after the last paragraph of the wording: the note is both paragraphs. Add (see 13.13.2) after “available” in the first paragraph (to key the reader that “available” is a technical term).

Change “where it is possible to ensure” to “when” in the summary.

Approve AI with changes: 6-0-0.

AI-341/01 Primitive subprograms are frozen with a tagged type

Pascal came up with a case where an instantiation of a generic subprogram with a tagged type occurred in the same scope as the tagged type. That would be a primitive routine of the tagged type; that is ugly. Tucker points out that you could still do that, even with this rule.

Pascal is worried about compatibility. But subprograms don’t have much that can be done with them (Import/Export/Convention, 'Address). Moving the offender would be easy in the case of a problem with this rule.

Approve AI: 8-0-1.

AI-349/01 Equality on private extensions

This is a ramification. The parent type in question is that of the full type.

Approve AI: 9-0-0.

AI-350/01 Allocating and comparing zero-size objects

This is a religious issue, and we can discuss it forever. We did, with no new information being presented. (Editor's Note: The interested reader is directed to the AI's appendix; reserve plenty of time to read it.)

Tucker moves No Action as this just isn't important to resolve (it's not causing real problems in real programs).

No Action: 5-3-1. (Pascal threatens to call for a letter ballot on this one.)

AI-373/02 Undefined discriminants caused by loose order of init requirements

Tucker thinks that in the first part of the replacement wording Steve is answering a question that wasn't asked, and it doesn't solve a problem. Steve disagrees, other changes force those changes.

The steps 3 and 4 in the original wording are defined to be sequential, so they have to be merged (because we certainly don't want them to be sequential).

The middle paragraph is a rewrite of the existing paragraph 20. Tucker says that "assignment" is an unnecessary change; Randy suggests that "evaluation" would be OK. Tucker still objects, saying they'd have to rewrite parts of their compiler to do this.

To handle this,

- Move the definition of *requires late initialization* earlier;
- Combine the remaining parts of the last two paragraphs, making this into four bullets;
- Leave the words as close as possible to the original 3.3.1(20).

There is a question about the tag; we should say that any implicit components are defined. Say "…any components, including implicit components, not requiring late initialization…".

"Assignments to" is wrong; subcomponents can assigned individually. Suggest: "The assignments to parts of any components, including implicit components, not requiring late initialization…" and then: "…then the assignments to parts of the component occurring earlier… initial value evaluations of the component occurring later."

No one really is sure what the wording is anymore.

We just vote to continue working on the AI: 7-0-0.

AI-376/01 Interfaces.C works for C++ as well

Approve AI: 3-0-3.

AI-377/00 Naming of generic child packages

Steve will handle this AI.

AI-378/01 The bounds of Ada.Exceptions.Exception_Name

Approve AI: 6-0-0.