Final Minutes of 8th ARG meeting
24-26 March 99
Burlington, USA

Attendance: Members: John Barnes, Randy Brukardt, Gary Dismukes, Bob Duff, Robert Eachus, Kiyoshi Ishihata, Mike Kamrad, Pascal Leroy, Erhard Ploedereder, Tucker Taft. Observer: Brian Nettleton (Aonix)

Meeting Summary

The meeting convened on 24 March at 0900 hours at the AverStar (formerly Intermetrics) Offices in Burlington MA and adjourned at 1515 hours on 26 March.

Most of the first two days were spent on AIs, part of the first morning on planning the Technical Corrigendum, and the last day on future extensions. The details on these topics are found in later sections of these minutes, which are not presented in chronological order.

The ARG unanimously thanked Tucker and AverStar for their generous hosting of the meeting.

AI Summary

The AIs were discussed in nearly the same order as listed in the meeting agenda and nearly the full agenda was covered. Only AIs without new write-ups since the last meeting were not discussed.

The following AI were approved without changes:

AI-147 - Optimization of Controlled Types (writeup revised at meeting) (6-0-2)
AI-175 - Full conformance of expressions with attributes (9-0-0)
AI-194 - Typo in Standard_Error Definition (9-0-0)
AI-198 - pragma Convention(Intrinsic) is not a completion (9-0-0)
AI-215 - Returning remote class-wide values (9-0-0)

The following AIs were approved with small changes, soon ready for editorial review:

AI-100 - Truncation required for static expressions if Machine_Rounds is false (8-0-2)
AI-117 - Calling Conventions (6-1-1)
AI-130 - Enforcing Restrictions might violate the generic contract model (8-0-1)
AI-131 - Interface to C -- passing records as parameters of mode 'in' (8-0-1)
AI-153 - Picture String Grammar or Composition Rules Need Tightening (8-0-2)
AI-169 - Exceptions raised by Adjust/Finalize -- Missing cases (8-0-1)
AI-170 - Can an attribute defined in an annex be set in an attribute definition clause? (8-0-1)
AI-172 - Main subprograms are optional (8-0-2)
AI-193 - Classwide <Initialize,> Adjust and exceptions (6-0-3)
AI-199 - Pragma convention for a generic unit applies to its instances (8-0-2)
AI-202 - Primitives of formal type derived from another formal type (9-0-1)
AI-207 - Pragma Inspection_Point affects only the current unit (10-0-0)
AI-208 - What is the meaning of "same representation" in all partitions? (9-0-0)
AI-210 - Questions on pragma Restriction No_Recursion and No_Reentrancy (8-1-1)
AI-211 - Can an abstract subprogram be renamed? (8-0-1)
AI-213 - Formal object matching for formal packages (9-0-0)

The following AIs were approved with changes, soon ready for the previously requested letter ballot:

AI-51 - Size and Alignment Clauses for Objects
AI-109 - Size and Alignment Attributes for Subtypes

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

AI-58 - Accessibility Rules for Shared_Passive Packages (8-0-1)
AI-161 - Default-initialized objects (9-0-0)
AI-186 - Range of root_integer (7-0-3)
AI-214 - Distinct Names for Compilation Units (again) (9-0-1)

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

AI-174 - Are 'Read and 'Write guaranteed to be "inverses" for predefined types?
AI-188 - The definition of setting a task base priority is too vague
AI-189 - The meaning of the terms "processor", "multiprocessor", and "processing node"
AI-195 - Streams
AI-204 - Interfaces.Fortran must be absent, right ?
AI-209 - pragma Reviewable; can objects become uninitialized?
AI-212 - Restrictions on configuration pragmas

The following AIs were not discussed and were assigned to editors for first versions:

AI-178 - Which I/O operations are potentially blocking?
AI-191 - An OBJECTive View
AI-196 - Assignment and tag-indeterminate calls with controlling results

The following AI was tabled pending resolution of AI-158:

AI-200 - Generic formal subprograms as dispatching operations

Technical Corrigendum

Randy explained the motivation behind each of the three prototype styles for the Corrigendum. The Corrigendum will be limited to the Binding Interpretation AIs (in addition to the editorial Presentation Issues) ; the other AIs are more usefully applied to the complete rewrite of the manual (sometime in the future). This launched a discussion of the real value of Corrigendum versus the raw content of the AIs. The vendors will likely only use the raw AIs, especially when they are accompanied by a set of tests. On the other hand, the Corrigendum is aimed at meeting an ISO requirement for collecting a listing of language changes; it could be a useful document to the Ada community to highlight the language changes. Consequently, the Corrigendum should be limited to just disclosing the paragraphs in the RM that are affected, the actual new wording, and a reference to the relevant AIs. This led to choosing Style 1 among the prototypes without the appendix as the generic format for the Technical Corrigendum. The content of the former appendix will be put into a separate document, under the ISO-required title of Defect Reports, which needs to accompany the Technical Corrigendum.

Next, we discussed how to complete the Corrigendum. Randy estimates that there are approximately 100 Binding Interpretations of which two-thirds have wording for the Corrigendum and the remaining third need wording. It was decided to divide the generation of Corrigendum wording equally among members of the ARG as assigned by Randy, with the possibility that the AIs may be swapped among ARG members. A new section, with !corrigendum heading, is where the wording will be added; it will appear just before the !appendix section. Randy will generate the !corrigendum sections for those AIs with existing wording. It is Randy's intent to then generate the Corrigendum semi-automatically from these !corrigendum sections. Paragraphs in the RM which are affected by multiple AIs will require manual intervention to resolve. Randy will cross-index the AIs that contribute to the Corrigendum by AI number and by RM paragraph number.

The schedule for completion is driven by the desire of the market place to have finished and published the Corrigendum in 2000, timed to correspond with the ISO 5-year milestone for reaffirmation of the standard. This means that first drafts of the wording are due by 1 July. The first draft of the Corrigendum would be reviewed at the next meeting this fall. It would then be updated and presented to WG9 at the June 2000 meeting for their review and then obtain ISO approval in Fall 2000. All draft wording should be reviewed internally as soon as possible; any

objections to wording must be accompanied with alternative wording.

Proposed enhancements will not be included in the Corrigendum. Instead, either an Amendment document (for new language features) or a Secondary Standard document (for new packages, pragmas, attributes and the like) will be used to collect the approved enhancements.

It was decided to not have a new document format for amendment/enhancement proposals. They will be written up in AI-format and included in the numbering of AIs. At most, we'll give them a status „enhancement".

Randy will make spelling corrections on his own.

ACATS Tests

Randy reminded the ARG of their intent to be more involved in conformance tests. New AIs require tests to effectively drive implementation. Therefore it was decided that the creation of the tests for Binding Interpretation AIs will be divided among members of the ARG, roughly along the lines of Corrigendum wording assignment. Bob Duff volunteered to assume a larger role in test creation in exchange for a reduced role in Corrigendum wording creation.

Next Meeting

The next meeting will be in Boston on 22-24 September (Note: in the meantime changed to 27-29 September) at the Top Layer facility in Westborough MA. Mike Kamrad will publish the logistics for the meeting soon.

Action Items

Many existing action items have been accomplished but old action items remain:

Norm Cohen: AI-133, AI-167, AI-173
Robert Dewar: AI-85 (distribute an "append" Test to comp.lang.ada and collate results to AI)
Gary Dismukes: AI-158
Robert Eachus: AI-185
Stephen Michell: AI-148, AI-187
Tucker Taft: AI-162

New action items on the drafting or completion of AIs were assigned to the following ARG members:

John Barnes: AI-204, AI-207
Randy Brukardt: AI-117, AI-131, AI-153, AI-170, AI-198, AI-208, AI-210, AI-211, AI-213, AI-218
Gary Dismukes: AI-169, AI-196, AI-202
Bob Duff: AI-51, AI-109
Bob Eachus: AI-100, AI-172, AI-174, AI-186
Mike Kamrad: AI-58
Pascal Leroy: AI-161, AI-178, AI-186 (with Bob E.), AI-195(1.part)
Erhard Ploedereder: AI-209, AI-212
Tucker Taft: AI-130, AI-188, AI-189, AI-191, AI-193, AI-195(2.part), AI-199, AI-214, AI-216, AI-217

The following action items were also assigned:

All: Create Corrigendum wording for assigned AIs
All: Create tests for assigned AIs
Bob Duff: Be the test creator of last resort
Randy Brukardt: Create the first draft of the Corrigendum
Mike Kamrad: Publish logistics for next meeting; produce a proposal for pragma Assert

Details of AI Review

AI-*

Randy will remove the dates in the AI numbering; these dates are no longer needed in the presence of the configuration management mechanisms that Randy has installed. Also, it will be Randy's responsibility to update the !status sections and to run a spell-check on the AIs.

AI-51/9

Tuck objects to the AI because it appears to require support for 4-byte integer objects in a stack frame to be aligned on an 8-byte boundary, if that is the maximum default alignment. His motivation is to remain consistent with the rules of existing code generators where they "naturally" want to select 4-byte boundary for integer objects. Tuck would like the AI to say that in the cases where the size is not specified, that the alignment be sensitive to the default size of the specific scalar type (not just the maximum default alignment of any type); implementation would not be required to support specified alignments larger than the default size.

This concern is limited to scalar objects. He favors little restrictions on records, except that the size should be a multiple of the alignment. As for the alignment of an array object, he favors requiring support only up to the alignment that is selected for its non-array subcomponent subtype; the non-array subcomponent type of the array subtype would be the first non-array component subtype that lies at the bottom of possibly nested array declarations.

There is not the same concern for statically allocated objects since this is governed by the capabilities of a linker, per RM 13.3(35).

Discussion was tabled until Bob could rewrite the fourth paragraph of the !summary to reflect this and to put it into a format similar to the third paragraph of the !summary. The rewritten paragraph 4 was presented and no changes were suggested.

The AI with changes (from Bob) will be distributed for letter ballot.

AI-58/7

This AI adds an erroneous execution condition to the Annex. Pascal sees this as a Binding Interpretation since a new semantic („not accessible“) is added. Unfortunately this means that new wording must be added, but continues to be hard to formulate (both when the feature was originally defined and now). If effective wording can be found then it should become a Binding Interpretation; otherwise it stays a Ramification, awaiting proper wording later. A straw-vote of 5-2-2 favored the Binding Interpretation status. Implementors will in any case be aware of this change because of its AI status.

The meeting approved (8-0-1) the intent of the AI and a search for wording to make it a Binding Interpretation. Mike will make an attempt at finding the wording.

AI-100/2

At first glance, for this special case dealing with machine_numbers, this is always true because the RM says so. But on closer inspection, this hand-waving won't work since machine_numbers don't apply to fixed numbers and so the summary is incorrectly applied to fixed point numbers. The fix is to define machine_numbers for fixed point numbers be multiples of ‘small. This makes this AI a Binding Interpretation.

This change is approved (8-0-2) and Robert Eachus will complete the rewrite by adding the question about the machine numbers of fixed point types and answering it.

AI-109/6

Since there is a direct correlation between this AI and AI-51, it was also tabled until Bob could present a replacement for the second paragraph of the Recommend Level of Support section of the !summary, essentially making AI-51 and AI-109 parallel statements in form and content. The rewritten paragraph was presented at the same time as the rewrite for AI-51. Several changes were recommended to direct the paragraph toward user-specified sizes and to

provide a better explanation of how to apply the rules for array subtypes (per the description in AI-51).

The AI with changes (from Bob) will be distributed for letter ballot.

AI-117/3

This issue was last discussed at the Burlington meeting in Spring 1998.

Pascal raises a concern about private types and conventions, because of the breach of privacy that arises because the convention of the full type determines the convention of the partial view of the type. This breach is acerbated, because interim declarations may in turn depend on the convention of the partial view. Consider:

```
   type T is private;
   type A is access procedure (X: T);
   pragma Convention (C, A);
private
   type T is …;
   pragma Convention (C, T);
```

The correspondence of Conventions is difficult to check in a single pass, as is the eligibility of T as a C-eligible type (e.g. T completed as a task type, which can't possibly be C-eligible). Randy and Tuck argue that it's not necessary to check the eligibility because anything is possible when interfacing to C (basically making the user responsible for the meaning of the parameter, e.g., a pass-through of a by-reference type might be quite okay). So don't worry about it, unless a second pass is added to test for this problem. Clearly, Pascal has a problem because of the violation of the privateness; unfortunately there is no alternative without severely undoing a variety of rules of the language (and AIs). See for example 13.1(9), which asserts that representational properties apply both to the partial and the full view of types. In essence, this is but another instance of a check that needs to be postponed until the respective type is completely defined.

There were several editorial changes: add „however“ and „result [legally] has“ to the last paragraph of !summary, delete note in !discussion about AI-65. In !wording, last para., change „inherited operation“ to „corresponding primitive operation of the parent type“.

Approved with editorial changes (6-1-1) and Randy will finish it.

AI-130/3

It was explained that the main change to this AI was the addition of the fourth bullet in the explanation of checks found in the !recommendation section. Because the !summary may be interpreted to require compile time checks, new words will be added to make sure that an implementation can perform these checks during post-compilation. In the !summary, change the term "logically expanded" to "logically appear" for clarity. It was noted that the !wording section needs more specific wording; this will be added to the !corrigendum section. It was further noted that, in the !discussion, the paragraph starting „to ease the burden...“ is nonsense, since the AI mandates post-compilation checks and not dynamic checks that are somehow inserted in the code. It needs to be restated in terms of post-compilation checks. In the !discussion, 2.line, „control“ should read „contract“.

Approved with these editorial changes (8-0-1) and Tuck will make the changes.

AI-131/7

Remove „and only if“ from paragraph 4 of !recommendations. Change reference from B.3 to B.1(13-18). Approved with editorial changes (8-0-1) and Randy will finish it.

AI-147/5

The email vote before the meeting was 5-1-2 prefering the inclusion of Initialize/Finalize pairs. Gary objects to this applying to Initialize/Finalize pairs because users must now make the type Limited_Controlled to guarantee the effect in the absence of a use of the initialized object.

Tuck objected to the wording in the summary. In particular, item a) in the !discussion fails to make it clear that it is the interaction of Initialization and Finalization pairs and not their contents that is of significance. Erhard agreed with this intent but tried to cast it in much shorter words in the !summary (as opposed to the luxury of the !discussion section that can be more easily understood). Erhard agreed with Tuck's point on item b) in the !discussion, which has to apply after the omission of calls; before the optimization it trivially applies.

The AI was approved (6-0-3) based on these changes. Later in the week Erhard presented a rewrite of the !discussion section and it was approved as is (6-0-2).

AI-153/1

Drop everything from the sentence starting with „However" to the end of the !discussion section as being irrelevant since the recommended phrasing is sufficient. Fix „{are} could" and „context[-]free" in the !discussion. Add a comma in the !wording before „or in".

Approved with editorial changes (8-0-2) and Randy will make changes.

AI-161/4

Change globally „base type" to „type" to eliminate out-of-date terminology. Also change the wording on the legal placement of the pragma (which must appear in the visible part, including the formal part, of a generic package or package) . In !wording, change „In the" to „If the".

The rules regarding types that are identified by this pragma are overly complex. So the discussion was devoted to finding ways to simplify the rules. In particular, in the !wording section, the following changes were recommended on the new wording to be inserted after RM95 10.2.1(11).

- Remove bullets 5 and 6; there doesn't seem to be a reason for the „preelaborable" requirement.
- Replace „subtype" with „type".
- Swap bullets 3 and 4 to avoid forward referencing.
- In bullet 2, rephrase to avoid the ambiguous binding of „with entry_declarations"; be more precise about the use of „overridden" (presumably „user-defined" is meant, inherited or otherwise); make sure, we're talking full type here.

There is a problem with derived types because the default expression on discriminants can be changed. Maybe the workaround is to force the user to specify the „preelaborable initialization-ness" of generic derived types. Alternatively, deal with the formal type as a partial view of the full type of the actual.

Fix the formatting in the list of affected packages. Also, this list needs appear in the !wording section and referred to in the !summary.

Considerable work is still needed to get the wording right. The meeting approved the direction of this AI (9-0-0), because there is no viable alternative. Pascal will do the rewrite.

AI-169/1

There was only one issue on the content; is it worth it to be this specific about raising an exception when the program is so badly broken that an exception is raised during finalization? The consensus was that it is useful to be this specific in order to guarantee that exceptions propagated by finalizations will not corrupt finalization of other types. Besides, the precedent has been set for this level of specification, it is one of the easiest to specify, it helps portable error recovery, and it's done already.

The subject line should change from "case" to "cases". In the !summary, the 3. para. should change from „should occur" to „occurs".

Approved with editorial changes (8-0-1) and Gary will make the changes.

AI-170/1

Extend the question in the !question section to ask for the case where the Annex doesn't explicitly allow it; then answer the question with "no". Change „explicitely“ to „explicitly“ (several occurrences).

Approved with editorial changes (8-0-1). Randy will make the changes.

AI-172

The first and last paragraphs of summary will be a surprise to most users. The requirement was added to support shared passive partitions and seemed harmless to add in the Core. Randy argues that it is unnecessary work for implementors and ACVCs to test for this.

To prevent poor press about this and extra work for implementors and the ACVCs, it was decided to change the title and to reduce the summary to the first paragraph only, where the "must" is changed to "may".

Editorials: „program“ (in several places) -> „subprogram“. „with no“ -> „without a“.

Approved with editorial changes (8-0-2) and Robert Eachus will make the changes.

AI-174/0

Robert Eachus verifies that these attributes only apply to types and that they are indeed inverses of each other (even in the presence of subtypes). Robert will complete the AI.

AI-175/2

Approved (9-0-0) as is.

AI-186/2

This problem appears to be relevant only inside of generics. Discussion is limited to standard types because, for the non-standard numeric types, an implementation can impose the restriction to be not allowed as actuals to a generic. Primarily the modular types seem to be at issue.

Pascal says his question was motivated by the implementation of Discrete Random generic package in a portable manner, where 'Pos is a useful attribute.

The discussion converged on the position that a non-static conversion of universal integer may raise a constraint error based only on the range of the target type. Example:

Foo : U := T'Pos (T'Last);

This should work and the operations should be conversions of universal_integer values.

Many editorials: spell-check the AI; add a few „T“ to „type“ to make bindings clearer; change „erroneous“ to something else. The !wording on 3.4.9 is identical to what the manual already says.
A „static“ in the discussion should be „non-static“.

The intent of the AI is approved (7-0-3) and Robert Eachus will rewrite with assistance from Pascal.

AI-188/3

There were two issues of concern:

The first issue was the request for an „immediacy“ requirement for changing a task's priority, especially, in the presence of an underlying operating system that can't support that level of immediacy. This immediacy requirement is aimed at tasks which are inside a tight loop and won't reach an abortion completion point; thus, the current

definition in the RM may be too loose by relying on completion points.

The consensus was that an implementation should have permission to determine how immediate the priority changes should occur.

The second issue concerned the multiprocessor/multiprocess terminology. While the discussion reflected a desire for clarifying this terminology, no consensus was apparent and the issue should be answered in AI-189, explicitly created for this purpose. (The notion that an implementation that could not support the immediacy of priority changes would then be regarded as a multi-processor implementation did not find many friends.)

Tuck will rewrite the AI to reconcile these views and have Ted Baker review the AI before bringing it back to the full ARG.

AI-189

Tuck will produce a draft, including some points from AI-188.

AI-193

The !title should include „Initialization“.

In order to clarify whether finalization is done in the case of a failed initialization by aggregate assignment, add the phrase "if and only if the initialization is successful". Additionally, the last paragraph is hard to understand, specifically, the term "adjustments due to be performed". The term is used in the RM but it is not well-defined and Tuck tried to make sense of it by explaining that, for Adjust invoked as part of initialization, if it propagates an exception, then no other adjustment need be performed before Program_Error is raised. Also, while the summary proposes to relax the definition, it does not say how. The !wording section needs content.
In the !discussion, „scope is exited“ should read „master is left“.

Approved with editorial changes (6-0-3) and Tuck will make changes to reflect all the above.

AI-194/1

Approved (9-0-0) as is.

AI-195/4

The following changes were recommended for the !summary section:

- Item 2: Stream-oriented attributes were incorrectly grouped with representation attributes; they need a new category. Add the phrase for „for the purposes of legality rules“ to this item.
- Item 4: Replace "base type" with "base subtype"; also in discussion
- Item 9: Add (short) words to explain that this is allowing for internal buffering

The following changes were recommended for the !discussion section:

- Item 4: Add a full type declaration in the example
- Item 9: Change "pervert" to "perverse"

Add the old vote (Paris, 8-0-0, with changes) to the status.
Pascal will make these changes.

The AI then will be passed to Tucker for adding the capability to specify default stream sizes of types by means of attribute definition clauses.

AI-198/1

Outside of moving a comma in paragraph 2 of !response, approved as is (9-0-0).

AI-199/2

Tuck added last paragraphs in both !recommendation and !discussion sections of the AI.

The main issue raised about this AI was the need for all this specification since the RM makes support of these pragmas for generic units implementation-defined, anyhow. A binding interpretation on something implementation-defined is somewhat strange. The counter-argument is that the approach in the !recommendation should be the standard for all those implementations that do support it.

This issue was resolved by making the approach defined in the !recommendation Implementation Advice, not a requirement.

Approved with these editorial changes (8-0-2) and Tuck will make the changes.

AI-200/1

Postponed discussion until AI-158 is resolved, since there is an interaction and the current conclusion of AI-200 has a good possibility to diametrically contradict the solution that AI-158 might arrive at.

AI-202/2

Most of the changes recommended were editorial in nature. Wording changes are needed in the !recommendation to replace parent type with ancestor type to clarify the recursive nature of this AI. The last question should be rephrased to shorten its long answer. The comments in the example code in the !question, particularly the one at position (5), should be reworked to illustrate more clearly the issue at hand.

Approved with editorial changes (9-0-1) and Gary will make them.

AI-204/1

The spirited email discussion spilled into the discussion. Robert Dewar's point about the value of the Interface.COBOL package for handling COBOL files makes a compelling case for having the package even without the COBOL Convention pragma. So the discussion focused whether the Convention pragma and the Interface package are tightly coupled to each other for each supported language and whether either or both are/should be optional. There is conflicting language found in paragraphs B.2 (13) and B.5 (20) which makes optionality and independence confusing.

Despite what is stated in the RM, the members decided to determine what makes sense by (controversial) discussion and then by vote. They agreed (10-0-0) that providing any language interfacing support should be optional and they agreed (8-1-1) that, in the presence of support for a language, support for the Convention pragma and the Interface package are independent of each other. Of course, if either feature is supported, it must be supported as defined. This applies to the Convention pragma as well as to the packages, in which the declared types must indeed be eligible for the respective language and not just some Ada definitions that have the postulated operations, but bear no resemblance to the type representation in the language interfaced to.

The optionality of the support makes the AI a Binding Interpretation.
The title should be changed appropriately.

John will be guided thusly in the rewrite.

AI-207/1

The second paragraph in the !summary is dropped (along with the second sentence of the !recommendation section) because it opens the door to defining this capability for global variables, which is out of scope of the question.

At the end of discussion, change „is expected to“ by „might“.

The AI with the above changes was approved (10-0-0). John will update the AI.

AI-208/2

Randy went to heroic efforts to say something about semantic compatibility of types and to not eliminate the offending paragraph E.2(13), as directed by the vote of the last meeting. Unfortunately, the effort was largely wasted as it is dubious for the language to enforce exactly identical representation across partitions, even for homogeneous systems. After some discussion, the ARG arrived again at the conclusion to delete the paragraph in question .

The meeting asked Randy (5-0-4) to rewrite the AI so that it requires the deletion of E.2(13).

When the rewrite was presented later in the meeting, a paragraph in the !discussion section was deleted because it was unnecessary history that is distracting. It was approved with several editorial changes (9-0-0).

AI-209/1

This AI is to some extent a compiler implementation issue: how many instances of usages of undefined values does an implementation have to catch? The AI should say that at least it should catch the first instance and that the extent to which the remaining instances are caught should be determined by customer-vendor interaction, since the customer is probably more knowledgeable than the vendor or the ARG on his specific needs. In other words, how many more instances are identified is an implementation-dependent matter. The meeting approved by clear majority in a strawvote to increase the implementation dependencies of this AI.

Consequently, the attitude of this AI should be to not prevent overly pessimistic implementations of the feature. One suggestion is that "cannot" be replaced by "need not" in the 2. sentence of the 2. paragraph of the !summary section.

Also drop the 3. sentence of the 2. paragraph because it is covered by the 1. sentence.

Erhard will rewrite the AI.

AI-210/1

Get rid of the third paragraph of the !question because it is obvious and distracting.

Approved with editorial changes (8-1-1) and Randy will make them.

AI-211/1

This AI deals with where an abstract subprogram can be renamed. It is okay for an abstract subprogram to be renamed before the parent type is frozen because it creates two primitive operations that can then be both overridden in a type extension. Renaming an inherited abstract subprogram is a bad idea in the case of a non-abstract type extension, since there is no provision to also override the renamed subprogram with an actual implementation; it therefore becomes illegal for other reasons.

There is a concern about how such renaming affects generics, given that the parameter association for subprograms is defined in terms of renaming and calls of abstract subprograms need to be dispatching, while calls on generic formal subprograms are never dispatching by current rules. This needs to be kept in mind when formulating the !wording; there may need to be an addition rule involving instantiations of generics.

The answers to the questions in the !question section should be changed to "yes, no, illegal, illegal", respectively, due to the reasons given above. Approved with editorial changes (8-0-1) and Randy will make them.

AI-212/0

There were two opposing views. Tuck wants the configuration pragma attached to the actual compilation unit that really needs it as a good documentation convention („I can only live in a partition with such-and-such partition-wide property“). Unfortunately this requires a link-time check on all units of the partition whether or not they abide by this property. The opposing view wants the ability to characterize applications by a library for each partition and by using "stand alone" configuration pragmas that apply to all the units in the library. This provides a good configuration control convention. A middle ground was found where, for partition-wide configuration pragmas, the stand-alone restriction should be allowed (maybe also confirming pragmas at compilation level); for other configuration pragmas both alternatives need to be supported.

Erhard will write the AI accordingly.

AI-213/2

The effect of this AI is to fix a bug in the RM and to bring the RM into compliance with how almost all implementations are handling this issue. The changes that were recommended were directed at more clarity and precise placement of new wording in the RM. In the !recommendation section, the new wording is:

"The intent of the standard is that actuals are always used for the matching rule of 12.7(6). In addition, the intent is that formals denote their actuals for the purposes of the matching rule of 12.7(6)"

In the !wording section, the new wording is:

"Add this paragraph after 12.7(8):

For the purposes of matching, any actual parameter which is the name of a formal object is replaced by the formal object's actual expression (recursively)."

In the summary: delete everything after „recursively". Change „(for" by „(of".

Approved (9-0-0) with these editorial changes and Randy will make changes.

AI-214/1

The question section appears complex with too many negatives in it. The second sentence in the question should go to the !discussion section. „not illegal" should be replaced by „legal".

Of more importance is how to best define an implementation permission on when to detect conflicts between the name of a compilation unit and the name introduced by a body_stub, prior to post-compilation. In other words, is the presence of a stub already sufficient grounds for rejection or must the stub be already filled with a compiled subunit ? There is a problem with the term stub as used in the AI, which is not a defined unit in libraries, where only compilation unit and subunits exist. It seems that users should be able to get the diagnostics as early as possible, i.e., the RM should not prescribe a delay of the check, nor should it presumably mandate the early check, as this may run afoul of existing implementation strategies.

Tuck agreed to take responsibility to make changes with regard to terminology. He is asked to also focus on what units can be removed from the library by the compilation of another unit, and not only on what units can be rejected.

Some interaction with AI-143 was noted; AI-143 deals with the issue of equal names in a partition and renders it illegal, while leaving it implementation-defined, how this is enforced. The AI at-hand deals with a more specific rule for enforcing the restriction and for interpreting deletion semantics associated with recompilation.

The intent of the AI is approved (9-0-1); Tuck will do the rewrite.

AI-215/1

This AI covers an oversight and brings function results into compliance with how the Annex handles formal parameters (Note: Implementations should perform the check at the call site to simplify the function implementation).

Approved (9-0-0) as is.

AIs lacking editors:

Pascal takes AI-178, Tucker takes AI-191, Gary takes AI-196, and Erhard takes AI-212.

Details on Enhancement Proposals

AI-216: Handling Mutually Dependent Type Definitions That Cross Packages

The access typemark must refer to another withed type. (The AI fails to state that.)

What about the size (or other representational detail, such as alignment, Storage_Pool, Convention, and so on) of the „with access"-type, if multiple kinds of access types are supported ? Three choices: Use a default representation, or provide a specification mechanism (pragma?) within the context clause, or allow a rep.-clause later on. Generally, there is a problem of specifying characteristics of the access type. (Note: this is similar to a shared generics implementation that is parameterized with access types.)

Pragma vs. rep clause: rep clause becomes a confirmation and needs dotted name to boot – both is outside current semantics and syntax.

User-defined equality is another problem. We cannot use the generics crutch, i.e., use the predefined one, since there will be regions where both the user-defined equality and the defaulted predefined one become visible; this would create a Beaujolais effect.

Tuck now believes that the with access type is something different from an „incomplete access type"; because it is problematic to use 'Storage_Pool and 'Size on these types.

We then revisited the need for including the subtype mark for the designated type: history shows that it helps for producing intermediate languages of existing back-ends, most notably the .Class files of the JVM; also it allows for T'Class, i.e. class-wide access types. Gary couldn't explain how J-GNAT handles this.

(Note: Modula-3 handles a similar problem by first guessing the size and if the guess is wrong when the type is completed then recompilation is required.)

The whole issue of generic contract model needs analysis? No, since incomplete types are not allowed as actuals, incomplete access types cannot arise in generics.

There is value for defining access parameters of these types – don't restrict the parameter type. Allowed uses should be: as component type, parameter and result type, access parameters, objects and return values.

Now for something different, from Erhard: Create a separate „foreword" / „abstract" package specification where all pertinent information on the mutual dependent types, including specifying size, alignment, storage_pool, is given in a partial view. The clients can depend on these characteristics, the package that completes the type now knows how to support the incomplete type. Side benefit: it is all supportable with the „usual" compilation order model. It seems to work, but is maybe a bit ugly.

Now for something different, take two, from Tuck: Combine the foreword and completing spec into a pure/preelaborated specification that is permitted to be imported in a mutually dependent manner.

But then, we discussed yet another problem that makes these incomplete access types really, really nasty....

Private problem: clients use the incomplete access type. This has transitive effects, especially when user-defined operations are involved. T in the example below is considered incomplete until the full type declaration of Foo.A1 occurs. What are the effects on the use of T, especially since privateness should, in theory, mandate that the user of T should be unaware of the fact that Foo.a1 plays a role in its declaration ? What about passing T to a generic formal type ?

```
with type Foo.t1;
with type Foo.a1 is access Foo.t1;
package P1 is
   type T is private;
private
   type T is record
      X: Foo.a1;
    end record;
end P1;

with P1;
package B is
    X: P1.T; -- which operations ? which consequences for the X component ?
```

A possible, but quite devastating solution is that the import of incomplete types causes importers of the importing package to expect the worst of private types in the client and clients of types of P1 and so forth, unless the importing package also import the packages with the full definition of the access type. This model is unacceptable, because all private types would be thus „poisoned“, regardless of whether the incomplete access type is used, and largely unusable. Also, the world becomes intractable, as mutually dependent types may span a larger cycle of packages („which packages need to be imported to get operations?“).

Now for something different, take three, from Pascal: Why not prevent importing of access types, since it is the source of all the problems and instead insist on local, normal, access type declarations with an incomplete target subtype ? This would then require type conversions between the local access type designating incomplete types and the „real“ access type definition. Implicit conversions could be made possible between these types.

Permitting implicit conversions of semantically compatible access types is intriguing to several people, especially Erhard who sees this solving another nagging problem, which is a real turn-off for Ada OOP:

```
package A is
   type t is tagged…
   procedure p (x: access t);
   type class_t is access t'class;
end A;

package B is
   type nt is new t;
   type class_nt is access nt'class;
   -- procedure p (x: access nt);
end B;

o : B.class_nt;  -- someplace  (usually some component of some access-designated object)

A.p(o);  --  currently illegal and a mystery/big nuisance to people with OOP experience in other languages.
B.p(o); -- legal
```

The first call could obviously be allowed (and would be, if compatible access types were implicitly convertible), since the call will dispatch to the right implementation in any case. This illegality is tied to the problem that, when both declarations of p are visible and if both were considered matching, current overloading rules would diagnose an ambiguity. However, the latter can be easily solved by adding the necessary words to the overloading rules, i.e., ambiguity that involves only the „same“ dispatching operations is okay. (Minor glitch: redefinable defaults in overriding subprograms -- this would have to be excluded in the permission.)

The consensus is that this seems to have real value. Erhard is asked to write this up as an AI.

Back to the AI at hand: the discussion of the AI should explain that it is the user-defined operations on access types that create problems for the original proposal (even if the revised proposal avoids the problem - this is for posterity, tracking the evolution).

Tuck will rewrite the AI taking all this into account.

AI-217: Unchecked Union

We discussed several technical issues of the AI:

Why defaults for discriminants? Mainly so that the compiler does not believe in an immutable discriminant. C-code could certainly change the discriminant. Defaulting does provide a lazy safety method. Constraining example:

X : MyStr (12);
Y : MyStr;

X := (11, „…“);
Y := (11, „…“);

The first case is erroneous for these cases where the check is suppressed. Calling C is essentially like suppressing such checks. Not every call on C with such a record should be potentially erroneous. The second case is fine, as long as the size was large enough -- hence the allocation of unions with maximum size.

After lengthy discussion, a straw vote on having defaults was inconclusive (0-2-8).

We discovered problems with generics (IN OUT objects of such types or such types as actuals to formal private types); which operations are actually available on such formal types ? See problems with some operations below.

generic
    Type t is private; --
    X : in out T;
....Using membership test within the generic will not work, when instantiated with a unionized type.

Also, a restriction is needed so that the (actually missing) discriminant cannot be used to constrain the subtype of a component. Otherwise, serious Size problems would result. Other uses of the discriminant inside the record seem fine.

What about the use of controlled components? Leave it up to the implementation to determine how to handle those.

What about access discriminants? They seem okay and useful for initialization.

What about access subtypes to unions? Permit them but note that erroneous situations like the above are possible.

In toto, the erroneous cases should be enumerated and examples for erroneousness added to the AI.

What about membership tests? They are not allowed, because they imply reading the discriminants ? Enumerate all the cases of reading. Note the interaction with generics: We can’t make the predefined operations illegal without breaking the contract model. We can make them raise Program_Error at some cost to implementations, or we could just plainly claim erroneousness in the generics case.

What about block comparison (and other block-operations)? Gap filling, component reordering, internal dope, and tagged types differences make block comparison unreliable. Predefined equality, ‘Read, etc. should raise a Program_Error exception.

What about sizing differences between limited/unlimited? There should be none. Counter-example: for an array with such a component type, indexing will go terribly wrong on the C-side, if there are differences in size. We also discovered an interaction with generics that confirms that for the unchecked_union records the maximum size needs to be allocated in all cases.

What about interaction with private types? Useful to know what restrictions are present in the partial view, such as restrictions on reading discriminants or membership tests.

What about untagged derived types where the pragma is applied to the derived type ? This might actually be useful for conversion that alters representation from Ada-safe to „union"-unsafe; the reverse conversion would need to be illegal and an aggregate would be needed instead. Other interactions when „discriminants" are changed by C-code ?

Somewhere the text should point out that mutable unions would not necessarily be kept on the heap in Ada.

AI-218: Accidental Overloading When Overriding

There are two possibilities to solve the problem (the third possibility of cobbling up syntax using existing keywords was discarded):
- pragma (similar to Restriction) to do the job since pragmas work well in this situation of getting additional checking done by the compiler
- new keyword like „redefine" or „new <method>"

The pragma approach has problems of placement and misspelling of identifiers. Appearing late is not good either; close proximity would be a good style convention. If you misspell the identifier in the spec, you can also easily misspell it in the pragma. The pragma would have to apply only to the most recent preceding declaration (not, as other pragmas do, to all preceding overloaded declarations).
The other alternative is to introduce a new keyword like „redefine", mimicking the way Eiffel does it. Apart from the problem of introducing new reserved words, it would not be easy to make this upward compatible.

An excellent example of the problem addressed by the AI is the Finalize/Finalise confusion with British spelling.

Randy will rewrite this AI with the pragma solution.

Pragma Assert

Pragma Assert (provided by some implementations) is useful and Mike will propose it.

Other Enhancements

Mike reported from the recent RT-Workshop and language enhancements most asked for in that corner: CPU-Budgeting as a scheduling regime; mutable ceiling priorities on protected objects (to support mode changes), extensible protected types, and priority inheritance across partitions.