Minutes of the 28th ARG Meeting

19-20 June 2005

York, United Kingdom

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns (missed morning and last 1.5 hours on Monday), Kiyoshi Ishihata, Pascal Leroy, Steve Michell, Erhard Ploedereder (missed last 1.5 hours on Monday), Jean-Pierre Rosen (missed morning on Monday), Ed Schonberg (missed morning on Sunday), Tucker Taft, Tullio Vardanega.

Observers: Javier Miranda

Meeting Summary

The meeting convened on 19 June 2005 at 09:20 hours and adjourned at 19:30 hours on 20 June 2005. The meeting was held in a conference room at the Royal York Hotel in York, United Kingdom. The meeting covered the entire agenda as well as various off-agenda items.

AI Summary

The following AIs were approved with editorial changes:

AI-415/03 Presentation changes to the Standard (12-0-0)
AI-433/02 Examples in the Standard (12-0-0)

The corrections to the following approved AIs were approved with changes (as a group unless otherwise noted — 8-0-0):

AI-161/10 Default-initialized objects
AI-162/08 Anonymous access types and task termination/controlled type finalization
AI-217-6/14 Limited with clauses
AI-217-6/15 Limited with clauses (7-0-3)
AI-218-3/09 Accidental overloading when overriding
AI-224/11 pragma Unsuppress
AI-230/17 Generalized use of anonymous access types
AI-231/15 Access-to-constant parameters and null-excluding access subtypes
AI-243/03 Is a subunit of a subunit of L also a subunit of L?
AI-251/21 Abstract Interfaces to provide multiple inheritance
AI-254/13 Anonymous access to subprogram types
AI-262/08 Access to private units in the private part
AI-268/04 Rounding of real static expressions
AI-269/07 Generic formal objects can be static in the instance
AI-270/08 Stream item size control
AI-280/06 Allocation, deallocation, and use of objects after finalization
AI-286/10 Assert pragma
AI-297/13 Timing events
AI-302-3/13 Containers library (10-0-2)
AI-305/11 New pragma and additional restriction identifiers for real-time systems
AI-307/12 Execution-time clocks
AI-318-2/09 Limited and anonymous access return types (some changes: 12-0-0)
AI-326/10 Incomplete types
AI-345/09 Protected and task interfaces
AI-348/08 Null procedures
AI-351/08 Time operations (some changes: 10-0-1)
AI-354/09 Group execution-time budgets

AI-357/10 Support for Deadlines and Earliest Deadline First Scheduling (10-0-2)
AI-361/04 Raise with message
AI-363/09 Eliminating access subtype problems
AI-366/09 More liberal rules for Pure units
AI-373/06 Undefined discriminants caused by loose order of init requirements
AI-382/04 Current instance rule and anonymous access types
AI-387/04 Introduction to Amendment
AI-395/08 Various clarifications regarding 16- and 32-bit characters
AI-396/03 The "no hidden interfaces" rule
AI-397/05 Conformance and overriding for entries and protected operations
AI-399/03 Single task and protected objects implementing interfaces
AI-400/04 Wide_ and Wide_Wide_ images of identifiers
AI-401/04 Terminology for interfaces
AI-404/05 Not null and all in access parameters and types
AI-407/05 Terminology and semantics of prefixed views
AI-409/03 Conformance and overload resolution related to anon access types
AI-412/05 Subtypes of incomplete types; renamings of limited views
AI-414/03 pragma No_Return for overriding procedures
AI-416/08 Access results, accessibility, and return statements
AI-418/03 Vector norm
AI-419/04 Limitedness of derived types
AI-423/06 Renaming, null exclusion and formal objects
AI-424/02 List of language-defined units
AI-426/03 Language-defined routines returning abnormal and invalid values
AI-427/04 Default parameters and Calendar operations

Detailed Minutes

Meeting Minutes

Under AI-344, some text is in green for some reason. It should be in black.

Several people would like additional time to read them, so approval is postponed until Monday.

No further corrections are given on Monday.

Approve the minutes of the 27th meeting with change: 10-0-0.

Schedule of the Amendment

Randy has finished adding editorial and other corrections through Section 11. How long remaining? After discussion, we decide on 2 months, plus a month for vacations. So, October 1 is the likely completion date.

By Jim's schedule, there would then be 30 days for public review. SIGAda is on the 14th of November.

Most likely, the National Body Submission would occur in mid-November. That is the Amendment only. Ada Europe gets the AARM immediately before.

The version that the ARG will send to WG 9 should be considered as a draft NBS. That will be sent by the 1st of October. Of course, there will be a draft AARM at the same time.

Steve Michell asks how we are going to finish this document. In particular, how are we going to review and approve the remaining corrections? We are planning to do this by one or more e-mail ballots. That requires participation by everyone, especially returning the ballot and comments in a timely fashion.

Steve Michell thinks that we would have to have a meeting to make these approvals. Most ARG members don't have a travel budget for an additional meeting, and waiting until November is too late. So we agree to try the letter ballot approach.

Tucker would like to organize corrections by section, rather than AI. That seems like a good idea, the standard is stable enough now for that. Randy will try to do that for future corrections lists.

Next Meeting

We should vote on the remaining lists of corrections by letter ballot. That means that we don't need a meeting before submitting the draft NBS version to the ARG.

So we probably will not have a meeting before the SIGAda one. SIGAda takes place on November 18-20 in Atlanta. Steve Michell suggests meeting after WG9 at SIGAda.

ASIS

Jean-Pierre asks what happens if there is no ASIS RG? One possibility is that WG9 might assign it to ARG. Can we do this work if asked?

The ASIS RG seems to be non-functional. It is annoying that we don't have a New Work Item for this task.

We could allow ASIS people as observers on the ARG for working on such a standard. That would work similarly to the way that Matt Heaney was an observer during the development of the containers packages.

Could we run ASIS as a separate (sub)group? Pascal doesn't want the administrative overhead of a second subgroup.

Thanks

The ARG thanks our host, Ada Europe for the fine accommodations for the meeting.

Old Action Items

Randy is way behind on the AARM; he's only finished through section 11, not even finishing the first action item on his list.

John didn't re-review the automatically generated annexes, waiting for a more complete AARM.

Bob Duff didn't do an automated comparison tool — Pascal took that over (and found many minor differences between the Amendment and AARM, contributing to Randy's lateness).

Pascal didn't write the correction to AI-394.

Steve Michell didn't review 13; we drafted Erhard, John, and Tucker to do so at the last minute. Erhard asks who has the homework to make a wax effigy of Steve.

All other reviews and action items were completed.

New Action Items

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

John Barnes

- Re-review automatically generated annexes of AARM (K, L, P, index).

Randy Brukardt:

- Finish draft NBS of the Amendment and a matching draft of the AARM by October 1st.

Editorial changes only:

- AI-161
- AI-162
- AI-217-6
- AI-218-3

- AI-231
- AI-254
- AI-262
- AI-269
- AI-280
- AI-302-3
- AI-305
- AI-318-2
- AI-345
- AI-351
- AI-357
- AI-361
- AI-373
- AI-382
- AI-397
- AI-409
- AI-412
- AI-415
- AI-416
- AI-419
- AI-423
- AI-426
- AI-427
- AI-433

Bob Duff:

- Review the remainder of the draft AARM as it is finished.

Pascal Leroy:

- Write a correction to AI-394 to get rid of "apply in this Annex" wording in H.4 (see Miscellaneous Topics).
- Develop wording to fix the overriding problems of AI-251.

Tucker Taft:

- Review changes to clause 6.5, and propose corrections if necessary.
- Develop detailed wording for allocators in Pure units; see the discussion of AI-366.
- Develop wording for the dynamic semantics of "implemented by" subprograms and entries; see the discussion of AI-345.

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.

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Draft 12 Consolidated AARM. Paragraph numbers in other drafts may vary.

Corrections to Approved AIs

Unless otherwise noted, all corrections were approved as a group (including the changes to them noted below). The vote was 8-0-0.

AI-161/10 Default-initialized objects

Tucker has asked that the statement about tasks and protected objects be mentioned normatively in 10.2.1. The new sentence is the last one in 10.2.1(11.2/2). It should be bracketed as redundant.

Write this sentence in the other order to be clearer:

"A protected type with entry_declarations [and]{or} a task type never {has}[have] preelaborable initialization."

AI-162/08 Anonymous access types and task termination/controlled type finalization

Randy reports that the only change was to 7.6.1(13.1/2) to change "which" to "that" in two places.

Tucker says that he doesn't understand this wording at all. Randy points out that he suggested it during the Atlanta ARG meeting, as recorded in the minutes. So he doesn't understand the problem.

After discussion, we wonder why we need this wording. The case in question (that of a delay statement with an expression requiring finalization or task termination) seems like an unusual case.

Tucker suggests that we could say that a delay_statement is not a master, so the expression would then be the master; that would be enough to cover this special case.

Pascal says that if a reasonable answer falls out of the wording, we should say nothing. And the answer we have seems reasonable to him.

So we should drop the first two sentences of 7.6.1(13.1/2). Is this issue worth fixing as Tucker suggests? The group says no.

AI-217-6/14 Limited with clauses

Randy describes the changes made:

- "Implicit" dropped from 10.1.1(26/2) [three places], and "the declaration of" added to 10.1.1(26.1/2) "declarations are elaborated, not entities".
- 10.1.1(15/2) was changed to say "nonlimited_with_clause", as a renaming is not allowed in a limited_with_clause.
- The added bullets 10.1.2(20-22/2) were reorganized.
- 10.1.2(20/2) has an extra "or" at the end.
- 10.1.2(23/2) [Note 3] was changed to clarify that it only applies to nonlimited views.

The Note 3 change should be dropped. The rewording seems to suggest that a name used in a limited_with_clause cannot be used in an expanded name. Other rules prevent the other uses, so the note is OK as it was.

There were additional issues with the wording that Randy, Tucker, and Pascal had been unable to resolve before the meeting. Randy says that he and Tucker were getting close to a resolution.

Tucker tries to explain the problem. The old wording describes a limited view as a library_item. Tucker would like a limited view to be a view of a library package. How does "limited view" fit into "compilation unit", "library unit", "library_item"? The declaration of a limited view is an implicit library_item. That library_item is a compilation unit that has an empty context clause. The latter specifies the dependences between units.

Tucker says that there is no problem with bodies, because the wording says that only "declarations" are visible in with_clauses. A body is not a declaration.

Randy and Tucker will try to have a wording proposal tomorrow.

AI-217-6/15 Limited with clauses

Tucker presents his wording changes. [Editor's note: this version of the AI was created from the wording after the meeting. The corrections below are included.]

Paragraph 10.1.2(6) is the one that is missing its number. There are extra words in the version that Tucker e-mailed; here is what it should be:

"A library_item {(and the corresponding library unit)} is *named* in a with_clause if it is denoted by a library_unit_name in the with_clause. A library_item {(and the corresponding library unit)} is *mentioned* in a with_clause if it is named in the with_clause or if it is denoted by a prefix in the with_clause."

As previously noted, Randy had already changed 10.1.2(20-24) into three bullets, but in a different order than Tucker did. We approved that change previously.

The 10.2(6) change is missing the closing } in the version that Tucker e-mailed.

Approve wording with changes: 7-0-3.

AI-218-3/09 Accidental overloading when overriding

Randy says that he created a new subclause 8.3.1 to contain these rules, in keeping with our general principle to keep the syntax and semantics together. He also changed "point" to "place" in the rules at Bob Duff's suggestion.

Not all of the rules for overriding_indicators are found in this section; we decided last time that it would be too painful to move the task and protected type ones. There is an AARM note 8.3.1(7.c/2) to note where the other rules are. This seems too important to leave to the AARM alone; 8.3.1(7.c/2) should be a user note.

AI-224/11 pragma Unsuppress

"(see below)" was removed from 11.5(7.1/2), because it refers to adjacent sentences.

AI-230/17 Generalized use of anonymous access types

The wording of 4.6(24.11/2) was improved.

AI-231/15 Access-to-constant parameters and null-excluding access subtypes

Randy and Pascal realized that the terminology was neither defined nor consistently used. Changes were made to define and consistently use a term, in 3.2(7-8/2), 3.10(12/2), 3.10(13.1/2), 3.10(14/2), and 11.5(11/2). In addition, wording was added to 3.10(13.1/2) and 3.4(6/2) to ensure that a subtype defined without an explicit null_exclusion inherits the "excludes null" property from the named subtype. And the legality rule 3.10(14.1/2) needed to cover all of the places that a null_exclusion can occur.

Tucker prefers that 3.10(13.1/2) does not have a list of places. Just say "a null_exclusion in a construct specifies ...". "does" should be "excludes null". No one objects.

Similarly, Tucker would prefer to reword 3.10(14.1/2) as "A null_exclusion immediately followed by a subtype_mark is only allowed if the subtype_mark denotes an access subtype that does not exclude null." Pascal wonders if "immediately follows" includes a comment between. Pascal prefers the current wording.

Tucker then suggests replacing "whose subtype_mark" with "with a subtype_mark". That doesn't seem to work.

Erhard suggests "...is only allowed if the subtype_mark in a .... denotes an access subtype that does not exclude null." Tucker is happy now.

We turn to 3.2(8/2). Tucker suggests "....the set of values does not include...".

People would like to combine these two parts. Perhaps we could say "...its constraint and any applicable null_exclusion". But we don't have a term like "applicable" for null_exclusions. "...its constraint and any exclusion of the null value."

Change 3.10(15/2) to "An access type satisfies an exclusion of the null value if does not…"

AI-243/03 Is a subunit of a subunit of L also a subunit of L?

Tucker had asked that 10.1.3(8/2) be reworded, and so it was.

AI-251/21 Abstract Interfaces to provide multiple inheritance

Randy explains the changes: 8.3(26.1/2) was reworded to be more like 8.3(12.3). "Interface" was added next to "array" in 3.7(1/2) and 3.7(8/2), so that interfaces can't have discriminants. Interface types are supposed to be composite, so they were added to the list in 3.2(4/2).

Pascal had noted a question about 8.3(26.1). This paragraph seems to be talking about "overriding at birth". Is that right? Probably not; overriding can happen at many different points.

Adam Beneschan had sent an example:

```
package Pak1 is
   type I1 is interface;
   procedure Foo (X : in out I1) is null;
end Pak1;

package Pak2 is
   type Rec is tagged null record;
   procedure Foo (X : in Rec);
end Pak2;

with Pak1, Pak2;
package Pak3 is
   type Child is new Pak2.Rec and Pak1.I1 with record ...
   -- (A)
   -- procedure Foo (X : in out Child) is null; -- Foo1
   -- procedure Foo (X : in Child);             -- Foo2
   procedure Foo (X : in Child);                -- Foo3
end Pak3;
```

Is Foo3 legal or illegal? If Foo2 overrides Foo1, then Foo3 is legal. If not, Foo3 is illegal because it is not conformant with both Foo1 and Foo2.

The type declaration is illegal, because it fails 3.9.2(10). Foo2 overrides Foo1, but it is not subtype conformant. Returning to 8.3(26.1). What is the first sentence for? It's handling the generic case (see 8.3(29.s)). The parenthetical remark applies to all the sentences.

If you use inherited subprograms prematurely (say as the default of a record component), you can use the wrong parameter names. But the same body will be executed. Tucker claims that you can call subprograms with the wrong parameter names in Ada 95. (You can only see the original operation in the visible part if it is overridden in the private part.) So that doesn't seem terribly interesting.

Pascal shows an example where it matters:

```
package Pak1 is
   type I1 is interface;
   procedure Foo (X : in I1) is null;
end Pak1;

package Pak2 is
   type Rec is tagged null record;
   procedure Foo (Y : in Rec) is null;
end Pak2;

with Pak1, Pak2;
package Pak3 is
   type Child is new Pak2.Rec and Pak1.I1 with record ...
   -- (A)
   -- procedure Foo (X : in Child) is null;  -- Foo1
   -- procedure Foo (Y : in Child) is null;  -- Foo2
private
   procedure Foo (Z : in Child); -- Foo3
end Pak3;
```

In the example above the clients of Pak3 can determine, by using calls with named notation, which of Foo1 and Foo2 overrode the other when Child was declared. Pascal suggests requiring that the overriding occur in the same list of declarations. That way, the parameter names are unambiguously defined by the explicit declaration Foo3.

Premature uses (in default expressions) could cause trouble with 8.3(12.3); one of the routines is arbitrarily chosen, and it could change from compilation to compilation. So a premature use could be visible and not portable.

Pascal will try to rewrite 8.3(26.1) to fix this.

On Monday, Pascal announces that he was supposed to write wording, but was unable to. He shows an example of where he got stuck.

```
      type I1 is interface;
      function F(X : I1) return Integer is abstract;

      type I2 is interface;
      function F(Y : I2) return Integer is abstract;

      type R is new I1 and I2 with ...;

      type A is access R'Class;
      B : A;

      ... := F (X => B.A); -- Default expression

      function F (Z : R) return Integer;
```

R inherits two Fs with parameter names X and Y respectively. These functions shall-be-overridden because R is not abstract, so they can be called (for instance in a default expression, taking care to avoid being caught by the freezing rules) before the place where they are finally overridden by the explicit declaration. Such a call can use named notation, so it can be used to determine which function was “arbitrarily chosen to override” the other. This is bad.

Tucker suggests that all of those routines be left visible if there is going to be overriding. But then that requires a two pass algorithm.

Tucker then suggests that if they conform fully, then you arbitrarily select one, otherwise you leave them all.

Pascal says that’s new. Tucker says that it already happens in generics. Pascal says that that never has worked in their compiler. OK, but that doesn’t seem like the best reason for language design.

The problem occurs only in functions. You can’t have a procedure here (they can’t appear in default expressions).

Randy suggests that if the homographs aren’t fully conformant, then it has to be overridden. So then it could be illegal to reference them.

Pascal says that it could be a name resolution rule. That would work, but that would cause Beaujolais-like effects. (Deleting a declaration could cause something unrelated to disappear — because some other declaration could become unambiguous.)

We definitely want the call to be illegal. So we really need a legality rule on it, because name resolution doesn’t work.

Something like: “If two homographs are implicitly declared at the same place, and the homographs are not fully conformant, the name that denotes such a declaration is illegal.”

In addition to that, we also need the rule that it is overridden in the same list of declarations. Tucker wonders if we still need that. Well, letting a user declare something in the visible part (overridden in the private part), that they can never call in a public way seems more like a bug than an intended feature.

Pascal will try to write wording for this; perhaps some of us can get together later this week to deal with it.

[Editor’s note: after the meeting Tucker and Pascal came up with wording which purports to address the above issues.]

AI-254/13 Anonymous access to subprogram types

Randy had corrected the accessibility level of these so that they can be passed to another anonymous access parameter.

Discussion of AI-416 showed that 3.10.2(13.1/2) is wrong. It should say “deeper” rather than “statically deeper”. A copy of the existing 3.10.2(13.1/2) should occur after 3.10.2(18).

AI-262/08 Access to private units in the private part

There were minor rewordings of 10.1.2(11-15/2) to fix the bullet format and clarify the requirements.

10.1.2(12/2) should end with a colon; the bullets should end with semicolons.

Someone complains that 10.1.2(12/2) is written as plural. This has to be plural, there should be an AARM note to explain. This applies only if *all* of the with_clauses that mention the name have “private”. Add “one or more” in front of with_clauses. So a note is optional (Randy still thinks it is a good idea).

AI-268/04 Rounding of real static expressions

The change of AI-263 needed be made to the rounding rules, so that generic formal derived types and generic formal scalar are on an equal footing.

Tucker disagrees with the change. He says that we know the base range for a formal derived type (if it is not derived from a formal type, which is covered). Steve Baird says that a fixed point type allowed changing the *small*; in that case, you wouldn't know the *small*, and the rounding depends on it.

The change is "safe", in that it doesn't break anything even if it proves to be unnecessary. The difference between the rules would be weird. So we will keep the change. Tucker says he will look at it.

AI-269/07 Generic formal objects can be static in the instance

The (old) bullet 4.9(37) was merged into 4.9(36/2) because it made things confusing.

4.9(36/2) should leave "small" in italics. It's an intentional use of italics, somewhat similar to that of universal.

AI-270/08 Stream item size control

The recommended level of support for Stream_Size had required being able to stream 40-bit float types (on typical machines). That doesn't seem sensible or useful, so float and access types were excepted from the requirements.

Tucker wonders if streaming odd sizes of integer types is useful. It's important for interchange, which is what this attribute is all about. Tucker would like a note saying that this does require streaming 24 bit values.

AI-280/06 Allocation, deallocation, and use of objects after finalization

Randy reports that the wording of 4.8(10.2/2) was changed slightly ("the" changed to "an"). 4.8(10.3/2) was reworded undefined "finished waiting". Added some missing words into 11.5(19.2/2).

11.5(19.2/2) needs to get rid of "finished waiting" as well. Use the same wording as 4.8(10.3/2) for it.

Tucker notes that 4.8(10.3/2) should say "the object to be created...". Similarly, in 11.5(19.2/2) we should say "tasks to be created".

AI-286/10 Assert pragma

There were no Name Resolution Rules for this pragma. Changed from "Boolean" to "boolean" so as to allow any boolean type, as in an if statement. The last paragraph of the Static Semantics was turned around. The first paragraph of the Dynamic Semantics had a number of minor changes. The note also had minor changes.

AI-297/13 Timing events

As discussed in Burlington, the type Timing_Event has been made tagged, and an AARM note added to explain why.

AI-302-3/13 Containers library

Randy read the list of changes made to the containers:

- Added Swap to the list of operations that cause a bounded error when reading an empty element.
- Added IA as requested in Burlington; also added wording to say that propagating an exception shouldn't drop elements.
- Replace_Element was missing from the "tamper with elements" list for sets.
- John noticed that the type of Length for Lists returned Natural rather than Count_Type.

There also are a large number of unresolved issues that we need to consider. John had a list from his work on the Rationale, and many others have gotten into the act as well.

John's suggestions are contained in http://www.ada-auth.org/ai-files/grab_bag/Contable.pdf.

John's item 1 suggested reordering the declarations, as described in the above file. There is general agreement that this is a good idea.

John's item 2 suggested adding Equivalent_Keys to Ordered_Maps and Equivalent_Elements to Ordered_Sets to match the formals of the hashed versions. Tucker says that this would not help changing from one to the other, as the formal names aren't generally visible. Pascal says it is useful to make these routines available, as they are part of the model.

John's item 3 was previously handled. There is a more general issue to be discussed later that subsumes John's item 4.

John's item 5 was a request to add the missing Insert for Vector that gives a default value for the new item. There is no objection to this change.

John's item 6 is to change Reverse_List to Reverse, and add a similar routine to Vector. This is a problem; **reverse** is a reserved word. So call these routines "Reverse_Container"? "Mirror"? "Flop"? "Invert"? "Reversal"? "Turn_Around"? Erhard suggests "Reverze"? (a joke??).

After the meeting, John and Pascal came up with Reverse_Elements as the name for these routines

There is a more general issue to be discussed later that subsumes John's item 7.

John's item 8 is that the formal type Key_Type of Generic_Keys should not be limited. Tucker asks if that is a feature. Not really, it's a part of the element (which is non-limited), and Update_Element_Preserving_Key requires making a copy of the key.

John's item 9 is another change to Generic_Keys. The ordering of the keys is given by two functions, "<" and ">" on keys and element. John thinks it would be better to give just one "<" on a pair of keys; this is much more likely to be available and it would greatly reduce the chances of error. The group agrees with this analysis.

.John's item 10 notes odd parameter names for Floor and Ceiling in Generic_Keys on Ordered Sets. These are "Item" rather than "Key". This appears to be a bug, and should be fixed.

Approve the changes discussed to this point: 10-0-1.

Randy describes an e-mail discussion that showed a design problem with the containers. Some routines allow modifying a container with only a cursor. That means that a constant (or **in** parameter) container can be modified simply by extracting an appropriate cursor and then modifying the object. That seems wrong.

The solution that the e-mail discussion arrived at is that any operation that modifies a container should have an **in out** container parameter (even if that is not logically needed by the operation). Only a small number of operations (Replace_Element, Update_Element, and Swap) are affected.

The group agrees with this change.

Another issue that came up was whether the intent is that ":=" on a container makes a deep copy. We probably need an implementation requirement to this effect if so. The group thinks that containers should have value semantics ("=" already does), and the wording ought to reflect that. In this case, Assign for Vectors is redundant, and should be removed.

John had previously asked the subtype Index_Subtype be removed from Vectors, since it isn't used for anything. Tucker says that he might have defended it, but he doesn't. So it is removed.

John and Pascal had a long running discussion about the definition of the equivalence relationship for Maps and Sets. That eventually led to rewording the relevant parts. We agree that the proposed wording is fine.

John had noted that Delete from Vectors and Lists has a Position parameter with mode **in out**, but the text doesn't say what it is set to. It should be set to No_Element.

An Ada Comment discussion on Replace_Element for Sets had noted that the semantics were based on always making a change. This resulted in the element being dropped in some cases; an operation like Replace_Element should never change the cardinality of the Set — it is supposed to change the value of the element (usually a portion that doesn't participate in equivalence). So Randy suggests the following semantics:

"If Position equals No_Element, then Constraint_Error is propagated. If Position does not designate an element in Container, then Program_Error is propagated. If an element equivalent to By is already present in Container at a position other than Position, Program_Error is propagated. Otherwise, Replace_Element assigns By to the element designated by Position. Any exception raised by the assignment is propagated."

Ed thinks that this operation doesn't make sense. This is the only way to change an element given a cursor (as opposed to a value), as there is no Update_Element for a Set). And a set container isn't necessarily a mathematical set. Tucker suggests that the Position should be **in out**, because it might change from the insertion. Randy notes that others had suggested that as well, but he didn't think it was necessary. He was overruled.

[After the meeting, this issue was revisited, because the logical properties of the cursor are not changed by Replace_Element. - Editor]

Randy had noted that the standard requires predefined types to stream properly. But cursors are generally just wrapped access values, and those don't stream properly. In general, we could only stream a cursor if we could stream the container along with it. That seems silly. Alternatives including defining them to be abnormal when streamed in, or having 'Read and 'Write raise Program_Error. The group thinks that having them raise Program_Error is the simplest.

Pascal noticed leftover uses of "node" (which should be "element") in A.18.2(217/2), A.18.7(45/2), A.18.7(53/2), A.18.7(61/2), A.18.9(94/2), A.18.9(114/2). These should be changed. Ed suggests using "pair" rather than "node" for Map.

Matt noticed the Delete for sets (A.18.7(45)) says that No_Element has no effect, rather than raising Constraint_Error. That's just a bug that needs to be fixed.

Matt asked if the parameter name "By" (originally from Ada.Strings) is still correct for Replace_Element, as all of the others are now "Item" or "New_Item". We agree that "By" should be "New_Item".

John asks whether "Ada.Strings.Hash" should be "Ada.Strings.Fixed.Hash". In any case, it seems like there ought to be "Ada.Strings.Bounded.Hash". It would look like Ada.Text_IO.Bounded. And then we certainly would want "Ada.Strings.Fixed.Hash".

Tucker objects, saying that he doesn't want a semantic dependence on Fixed for hashing. Ada.Strings.Fixed has all kinds of junk in it. He suggests a renaming of the hash function for Fixed.

We should require that the same string generates the same hash value for all of these. Say "...is equivalent to the function call Ada.Strings.Hash (To_String (Key));"

John wonders why the Generic_Keys exports a lot of stuff on cursors vs. keys, but there is no cursor vs. cursor operation. These operations are provided by the main package; there is no need to replicate them in Generic_Keys.

Steve Baird asks what "=" means on a cursor? It should return true if both are No_Element or both designate the same element in the same container. These also have to compose (by the normal rules for predefined types), so implementers have to be careful. Wording like "The predefined "=" operator for type Cursor should return True if both cursors or No_Element, or designate the same element in the same container." needs to be added to each container.

Randy and Steve worry about cases where the cursors don't match naturally, so you have to make them tagged. Ugh. Tucker can't think of any legitimate case for that. Randy and Steve both give examples that prove to not have problems. Then Steve gives another example. Pascal tries to give an example, but he too fails. Thus, it seems that there is no real problem; natural matching will work.

Randy asks if we've given up on the idea that there isn't necessarily a single value for No_Element. In that case, natural matching wouldn't work. Well, we haven't really given up on that, but it doesn't seem worth worrying about.

Approve changes: 10-0-2.

AI-305/11 New pragma and additional restriction identifiers for real-time systems

Added a note to 9.5.1 to note the existence of pragma Detect_Blocking.

Tucker says that the note should say "...all executions of potentially blocking operations during a protected action raise...".

AI-307/12 Execution-time clocks

As discussed in Burlington, the type Timing has been made tagged, and an AARM note added to explain why. Also, a default of Time_Span_Zero for the TS parameter of Time_Of was added.

AI-318-2/09 Limited and anonymous access return types

Randy says that a lot of wording was adjusted for this one.

- The wording of 6.5(5.2/2) is inconsistent with surrounding paragraphs, and, to a lesser extent, so is 6.5(5.3/2) and 6.5(5.4/2).
- Pascal pointed out that an extended_return_statement is a compound_statement (since it contains statements), so the definition of return_statement doesn't work. Defined "return statement" to replace the syntax production, and changed the wording of the entire standard to match (dozens of places).
- 7.6(17.1/2) conflicted with 7.5(8/2). (7.6(17.1/2) exempted all return statements from the aggregate build-in-place requirements.) This should not apply to extended_return_statements (which are mostly like an object_declaration, and thus should be built-in-place).
- 4.3.3(11/2) needs to talk about the subtype of the result object, not the function result type. (They're not necessarily the same for an extended_return_statement.)
- 6.5(3/2) was changed to say that only simple returns have return expressions; it's confusing to say that the default expression of an extended_return_statement is a return expression (since it may not actually be returned. This forces various other changes (to 6.5(5.1/2), 6.5(5.3/2), 6.5(5.6/2)).
- 3.9(24/2) says that the result tag of a class-wide function is that of the "return expression"; but there isn't necessarily one (extended_return_statements don't have them). Changed to "return object", which seems better anyway.
- 6.5(5.4/2) needs to say "the" nominal subtype.
- The Note 7.3(19) is now wrong; it was revised.
- In 6.3.1(16), the first "and" was unintentionally deleted.

6.5(5.6/2) should say "...; the converted value is assigned to the return object. Otherwise,..." Similarly, 6.5(6/2) should say "...{,}[and]...and assigned to the anonymous return object."

Tucker would like to drop return statement altogether from 7.6(17.1); that would require build in place for the anonymous object (but not the result location).

Tucker hates the change to "return expression". He believes that there is an equivalence between the two kinds of return expressions.

Jean-Pierre and others say that return expression is just the wrong term for extended return — it's just an initial value. There also doesn't necessarily have to be an expression, either, and we need to make a check even if there is no expression.

Tucker will review these changes, and try to fix things if broken.

Bob had made a comment on 6.5(22) which hadn't been successfully resolved. The wording says that a return statement causes "return to the caller". But when a simple_return_statement applies to an extended_return_statement, it doesn't actually return; it's the completion of the outer return that does that.

The first suggestion is "Finally, a transfer of control is performed which completes the execution of the construct to which the return statement applies, and returns to the caller."

Tucker suggests: "Upon execution of a simple_return_statement that applies to an extended_return_statement, the extended_return_statement is completed. Upon completion of an extended_return_statement or a simple_return_statement that applies to a callable construct, a transfer of control is performed which completes the execution of the callable construct, and returns to the caller."

Tucker now is wondering where the check of 6.5(20.1) is made. For an extended_return_statement, we would want this done before the execution of the handled_sequence_of_statements.

Move the last sentence of 6.5(5.6) to become the first sentence of 6.5(22) with suitable rewording.

"For the execution of an extended_return_statement, the handled_sequenence_of_statements is executed." Randy would prefer "then" or something to indicate the temporal ordering. Tucker doesn't think that is necessary.

The complete new 6.5(22) reads:

"For the execution of an extended_return_statement, the handled_sequence_of_statements is executed. Within this handled_sequence_of_statements, the execution of a simple_return_statement that applies to the extended_return_statement causes a transfer of control that completes the extended_return_statement. Upon completion of a return statement that applies to a callable construct, a transfer of control is performed which completes the execution of the callable construct, and returns to the caller."

Approve changes: 12-0-0.

AI-326/10 Incomplete types

Changed "may" to "shall" in the last sentence of 3.9.1(9.3/2) so the AARM and AI match.

Simplified 10.1.1(12.3/2).

AI-345/09 Protected and task interfaces

The definition of dispatching in 3.9.2(20) did not cover "implemented by"; it was changed to fix that.

The wording "implemented entry or operation" is weird; task entries don't have bodies. So what is being executed? We need a wrapper model for implemented entries? No, not in general, as that breaks timed entry calls. What is the dynamic model of "implemented by" anyway?

Tucker will take this offline and try to fix 3.9.2(20) and the missing dynamic parts of 9.1 and 9.4.

The Note 3.4(35.2) was added.

The Note 3.2(12-3) was reorganized as Tucker has suggested in e-mail.

3.2(12): Erhard complains that "other limited interfaces" are not covered. The list appears exhaustive. If we don't add that, then we should get rid of "other enumeration" and "other array". After discussion, leave as is.

Added the new sentence of 9.7.2(1) to 9.7.3(1) in order to reinforce that timed and conditional entry calls have the same rules.

9.7.2(4) was reworded. Tucker asks that the last ", and" be changed to a ";". Steve Michell asks that a ";" be added instead of the first comma. No, that would be wrong.

We start discussing 9.7.2(3.2), which we had discussed in e-mail. In particular, should it be allowed to call a renamed subprogram in a timed entry?

Erhard objects to the entire notion of not having blocking specified visibly (that is, in the syntax somehow). That seems be out of bounds at this point; the model was known by WG 9 when they approved this AI.

We list the possible resolutions to what is allowed in a timed/conditional entry call:

1) any procedure or entry call;

2) entry call, primitive procedure of limited interface, formal subprograms, or renaming of one of the former;

3) entry call, primitive procedure of limited interface, or renaming of one of the former;

4) entry call or primitive procedure of limited interface;

5) any procedure or entry call with a runtime check that the routine is really an entry.

Erhard says that the runtime check is silly; the user expects that an exception is an error, but here there is a good model that produces the right answer had the call been allowed. No one other than Pascal seems interested in the runtime check, so we discard option 5.

Not allowing renames (option 4) when it is possible to see that the operation is really an entry is also silly. Moreover, we want renames to be semantically neutral. We discard option 4.

Aren't formal subprograms supposed to be the same as a rename? Yes, but this is a case where we can't do that for contract model reasons; we either have to allow *all* formal subprograms or *no* formal subprograms in bodies — otherwise, the bodies legality would depend on the actual parameters of the instance. That's just a standard assume-the-worst rule. We don't need to worry about specifications as a timed or conditional entry call can't appear there anyway. This is different than a regular rename, where inspecting the rename itself (perhaps recursively) will make it obvious if it is renaming an entry or a might-be-implemented-by-an-entry procedure.

If we're going to allow all of these cases (including some regular procedures) anyway, why should we have any legality rule at all? Jean-Pierre says that making a timed entry call and especially an ATC on a procedure is a mistake; the programmer might be expecting the procedure to be aborted if it doesn't complete in time, while it is really the start of the procedure that is being timed (which is always ready, so nothing useful happens).

Having no clear conclusion so far, we take a "can live with" poll. Option 1: 5-7-0; Option 2: 5-6-1; Option 3: 8-2-2

We decide to adopt option 3. This is a change; we're dropping formal subprograms from the types of entities allowed.

AI-348/08 Null procedures

In order to make 6.1 consistent with our normal practice, moved the syntax for abstract_subprogram_declaration to 3.9.3 to be with the semantics. Modified 3.9.3(3/2), 6.1(30/2), and moved the abstract part of 6.1(31/2) to make that consistent.

Added null procedures to (20). Added a rule stating that the elaboration of a null_procedure_declaration has no effect.

Added overriding_indicator to the syntax for null procedure.

6.1(20) boldfaces **pragma** instead of using the syntax font.

Bob told us that syntax is organized top down (this was news to everyone, we thought it was random), thus the new productions were moved after 6.1(3).

AI-351/08 Time operations

Randy explains the changes:

- Changed the wording of Split for Seconds and Seconds_Of.
- The characters in the description of Image were replaced with their names.
- The first note was replaced with an AARM note, as the note wasn't strictly true, and the full explanation is too long.

Jean-Pierre asks that the characters in the string in 9.6.1(82) be specified somehow. Randy says that Bob had suggested that this be set in the fixed width font. That seems too weird. Pascal says that a formal description would be incomprehensible. Jean-Pierre suggests "The separators between the values are a minus, another minus, a colon, and a single space between the Day and Hour."

Tucker would like: "...where the Year is a 4 digit value, and all others are 2-digit values, of ...".

Jean-Pierre suggests "...a point followed by a 2-digit value".

Similar changes are required in 9.6.1(86).

Bob wants to change the Leap_Second parameters to have an integer type (Leap_Second : range 0 .. 1) as above. Robert Dewar agreed, Robert Eachus opposed.

Erhard prefers no change. Tucker doesn't understand the point of the change.

Tucker would prefer to change "not appropriate" to "do not correspond to a second immediately preceding a leap second" in 9.6.1(74) and 9.6.1(76).

We take a straw vote: Boolean: 10; Integer range 0..1: 1; Abstain: 1

So no change in types of the parameters.

Erhard wonders if the wording includes the leap second. The intent is that "time values" includes the Leap_Second parameter.

Tucker thinks that more clarity would be helpful. He suggests: "If Leap_Second is False, returns a Time built... If Leap_Second is True, returns the Time that represents the time within the leap second that is one second later than the time specified by the other parameters; if there is no such leap second, Time_Error is propagated."

We're missing the rules that allow raising Time_Error for bogus dates like February 30.

Drop the last part of Tucker's wording, and then add: "Time_Error is raised if the parameters do not form a proper date or time."

This needs to be done for both Time_Of functions.

The "second before the leap second" wording needs to be added to Split.

Approve leap second changes: 10-0-1.

AI-354/09 Group execution-time budgets

As discussed in Burlington, the type Timing has been made tagged, and an AARM note added to explain why.

AI-357/10 Support for Deadlines and Earliest Deadline First Scheduling

[Editor's note: The wording corrections in this version of the AI were decided upon during the meeting.]

Alan and Erhard have been battling about the wording of this in e-mail.

After a couple of false starts, Erhard says that D.2.6(24) is the main disagreement. This rule says that the task goes to the ready queue of the lowest priority; there is no mention of the active priority.

Alan says that the ceiling locking priority says that the active priority is changed. This could happen in a rendezvous.

Another concern of Erhard's is that there is not always an active priority. Every other policy always has an active priority. He thinks there are various rules which expect an active priority. Alan says that D.2.6(26) defines the active priority; it depends on what queue it came from. Tucker suggests that the active priority defines the queue that the task goes onto, rather than the existing wording (which is the other way around).

Erhard mentions that preemption levels (who am I allowed to preempt) map onto base priorities, in this model. The only purpose of base priorities in this model is to define a preemption level.

Pascal agrees with Erhard; active priority is defined everywhere in Annex D.

Alan says that if the task is blocked, the active priority doesn't matter. Moreover, it changes frequently. But that seems OK; no one would calculate the active priority when the task is blocked.

Tucker suggests that the active priority when you are blocked should be the ready queue that you are on. That's the model of the other scheduling policies.

Erhard notes that the active priority is the maximum of the lowest of the range and any ceiling priorities. Better to say any inherited priorities (instead of ceiling priorities).

So Erhard suggests that paragraph 24 simply say that it is added to the queue "for its active priority". Paragraphs 20-23 would serve to define the active priority of the task.

Erhard's suggested wording doesn't work, because paragraph 26 says that the active priority is sticky; it doesn't change while the task is running.

Alan thinks it is silly to try to have an active priority all the time. Tucker says maybe we need to fix the general definitions to drop that as a requirement. Pascal says that's ugly and possibly going to cause trouble with other policies.

Try to define wording that defines active priority at all times; and has the same semantics as the current draft of the AI. This will be taken off-line.

Erhard says that paragraph 23 needs to say that the active priority is taken into account; we don't want this to lower priority. Alan thinks that happens only if there is an ordering problem. Someone adds "or if there is a bounded error".

Tucker will try rewording this.

On Monday, Tucker presents his improved wording, which is now version 10 of the AI.

The base priority level indicates the preemption level for this task.

Jean-Pierre asks if ceiling is checked with the active priority or base priority. Alan replies, yes, if you get the priorities right (the ceiling priority should be set based on the base priority — the active priority is always less). If you get it wrong, weird things happen. It isn't any worse than the other policies. And it can be checked statically (comparing ceilings to base priorities).

Tucker says that unblocking a task isn't covered by the wording — D.2.6(15-8).

Reword D.2.6(17). Erhard had suggested "There is a non-empty ready queue, ...". Alan prefers "There is a task on the ready queue for the active priority of T with a deadline earlier than the deadline of T;".

Steve Baird asks if Relative_Deadline requires referencing expressions that haven't been elaborated yet. D.2.6(10) specifies exactly when the expression gets evaluated.

Approve changes: 10-0-2.

AI-361/04 Raise with message

The wording of 11.3(4) and 11.4.1(10) were clarified based on suggestions from reviewers.

In 11.3(4), drop “a call of” since it says it twice: “a call of a subsequent call of”. Shouldn’t have “Ada.” on the name; “Exception” should be “Exceptions”. Tucker suggests instead “if a string expression is present, the value of the expression is associated with the exception occurrence.” That’s better, it avoids double definitions.

There is something goofy in AARM 11.4.1(10.f), it has a strange font size.

AI-363/09 Eliminating access subtype problems

The note 5.2(16) is not longer true; it was deleted.

AI-366/09 More liberal rules for Pure units

An incompatibility was eliminated by changing 10.2.1(17) to apply only to nonlimited partial views, so it *does* apply to private extensions, and does *not* apply to limited private types, to eliminate the incompatibility. Limited types aren’t externally streamable anyway (unless the attributes are defined explicitly), so this change doesn’t lose any functionality.

There is an open problem with 10.2.1(16).

The primary issue is that “normal” anonymous access types have storage pools; these shouldn’t be allowed in a pure unit. (Actually, all anonymous access types have pools, but for access discriminants and access parameters, the pool can be declared at the point of use, not the point of declaration.) We had handled that by requiring all access types to have Storage_Size specified to be 0. But you can’t specify Storage_Size = 0 for anonymous access types, so that’s limiting.

Pascal had suggested disallowing allocators of these types instead of banning the types.

There is also an existing problem with allocators of anonymous access types. That requires a separate rule for access discriminants. (We don’t want any allocations from a pool to happen; that could happen in a subtype declaration.)

Steve Baird asks if there is a problem for generic formal arrays. After some discussion, Tucker writes an example:

```
package P is
   pragma Pure (P);
   type A is array (1.10) of access Integer;
end P;

generic
   type T is array (1..10) of access Integer;
package Gen is
   Obj : T := (others => new Integer'(10)); -- Ouch!
   ...
end Gen;
```

We don’t want the type declared in package P to have any state (that is, a runtime pool), so the allocator must fail. How to accomplish that? We list the options:

1) Disallow anonymous access types in library level pure units.

2) Disallow allocators when we know:

A) Storage_Size when used in generics;

B) Disallow use as actual parameter to generic.

Steve Baird notes that 2A is similar to accessibility itself. 2B has lots of dragons (privacy). Any static rule might as well be rechecked in the private part.

We turn to an example of the other issue:

```
package P2 is
   pragma Pure (P);
   type B (D : access Integer := new Integer'(2)) is ...
   subtype BS is B (new Integer'(3)); -- This needs to be illegal.
   X : constant B := (others => <>);
end P2;
```

Steve Baird asks about a constant (see X above). We need to disallow allocator to variable in elaboration of pure unit.

Do we need a set of messy rule like preelaboration has for generic bodies? It seems like it.

```
generic
   type B is private;
package PG is ...
   pragma Pure (PG);
end PG;

package body PG is
   X : constant array (1..1) of B := (others => <>);
end PG;
```

The problem is the <>; it could hide an expression not allowed in a pure unit. The ancestor of an extension aggregate (Parent **with null record**); has the same problem.

Therefore, for generic bodies, formal private types and private extensions do not have pure default initialization. The rule also has to apply to (non-formal) private types and private extensions.

Approve intent of change: 11-0-1.

Tucker will create the wording changes needed to implement this resolution.

AI-373/06 Undefined discriminants caused by loose order of init requirements

Added the definition of the term "initialized by default" to 3.3.1(18). The term is already used in many places (4.3.1, 4.3.2 (in Ada 95!), 4.3.3, 6.5), and it is better to have it defined.

Rewrote 7.6(10) to say that Initialize is called on all objects that are "initialized by default". That's so all of the new cases are covered as well as the old ones.

Similarly, rewrote 4.8(7-10) to use the magic words "initialized by default". That's needed so an allocator is initialized in the same order as stand-alone objects — all of the "requires late initialization" stuff should apply to allocated objects.

Tucker objects to the new wording of 4.8(10), saying that a subtype_indication is not a nominal subtype. So change "as its nominal subtype" to "to determine its nominal subtype"

Erhard thinks that 7.6(11) is awkward, and suggests "...ancestor_part is a subtype_mark denoting a controlled subtype, the Initialize..." instead.

AI-382/04 Current instance rule and anonymous access types

Randy corrected the note 9.1(19) to reflect the rule change of this AI.

Add a comma before “other”. But this wording is not liked. Turn the wording around:

“Other than in an access_definition, the name of a task unit within the declaration or body of the task unit denotes the current instance...”.

AI-387/04 Introduction to Amendment

The lists of sections for 16-bit and 32-bit characters were changed to be consistent; see paragraphs 45 and 57.4.

AI-395/08 Various clarifications regarding 16- and 32-bit characters

Randy reports two changes with this AI:

- A number of typos in A.3.1(6) were fixed: missing hyphens, extra “Ada.” prefix.
- 6.1(10) was changed again to make Bob happier.

AI-396/03 The “no hidden interfaces” rule

Because of 3.9.1(3) (no record extensions of synchronized tagged types), we need this property (synchronized tagged types) to be visible in partial views. So, a rule was added to require that property to match for partial views, along with any interfaces. Both rules were moved to 7.3(7), as that is where partial views are discussed.

AI-397/05 Conformance and overriding for entries and protected operations

We approved a correction in Burlington to add two legality rules, but didn’t specify the wording or location. That was added here. We also were missing dynamic semantics for “implemented by”. Finally, we needed a syntax legality rule like 9.5.3(10.1/2) to 10.1.1 in order to be consistent, as it never makes sense to have overriding_indicators on compilation units.

Tucker notes that “which” should be “that” in 10.1.1(8.1/2).

Tucker would prefer that this just says library_item. But that isn’t directly the item with the indicator; this is a syntax rule and the wording has to follow from the syntax.

Steve Michell suggests “An overriding_indicator is not allowed in a library unit that is declared by ...” Pascal and Tucker object to this wording, because it seems to refer to the library unit, not the syntax. We decide to use the original wording (other than changing to “that”).

As noted under the discussion for AI-218-3, a user note should be added to clause 8.3.1 to point out the existence and location of these rules.

AI-399/03 Single task and protected objects implementing interfaces

Much of the revised wording had referred to task_type_declaration rather than “task declaration” (the latter includes single_task_declaration). A similar fix was applied to protected declarations.

AI-400/04 Wide_ and Wide_Wide_ images of identifiers

Randy explained that 11.4.1(12) was fixed, because the wording that had been proposed was inconsistent with the wording of other parts of this clause.

AI-401/04 Terminology for interfaces

There were multiple changes in this AI:

- The note 3.4(35.1/2) was a lie; interfaces are not derived types.
- The wording of 3.4(5/2) was wrong. Neither “and” nor “or” give the right semantics, so it was split into two sentences.
- 3.4.1(2/2) needed to say that the ancestors of progenitors are derived from.
- "=" should be inheritable from progenitors for limited types, 3.4(17/2) was fixed accordingly.
- In 12.5.1(21/2), “the progenitor types” were changed to “any progenitor types”.

Tucker would prefer “or an optional interface_list...” in 3.4(5/2). No, it’s clearer the way it is, but it should be marked redundant, because it follows from the syntax and the previous sentence.

AI-404/05 Not null and all in access parameters and types

The wording of 3.9.2(11/2) was fixed to use “subtype that excludes null”.

AI-407/05 Terminology and semantics of prefixed views

Rewrote the example to use AI-433’s interface example as the root.

AI-409/03 Conformance and overload resolution related to anon access types

Bob asked for a rewrite of 8.5.1(3/2), but the result was confusing. Tucker suggests splitting at “which”; “. If the anonymous access type is an access-to-object type, the type of the object_name shall have... . If the anonymous access type is access-to-subprogram type...”

AI-412/05 Subtypes of incomplete types; renamings of limited views

Changed “nearest” to “innermost” in 8.5.3(3.1/2). In that paragraph, replace the parenthesis with commas.

AI-414/03 pragma No_Return for overriding procedures

Changed return_statement to return statement as required by AI-318-2.

AI-416/08 Access results, accessibility, and return statements

Randy says that there were many changes to this AI:

- The source of an assignment_statement was omitted from the list of things that give function calls an accessibility level; a change to 3.10.2(10/2) was made.

- There seems to be no remaining reason for a declaration to be a master. So 7.6.1(3/2) and 3.10.2(7/2) were adjusted appropriately.
- Changed "immediately enclosing master" to "innermost master" in 3.10.2(10/2) and 3.10.2(14/2) to match 3.10.2(7/2)).
- Added 3.10.2(16.1/2) to say that uses in a qualified_expression, etc. are the same as direct uses. This was added as a separate paragraph in order to be recursive and to apply to both 3.10.2(10/2) and 3.10.2(14/2).
- Changed 6.5(5.4/2) to be more similar to surrounding paragraphs by adding "of the function" after "result type".
- Changed 6.5(5.4/2, 20.1/2, and 23/2) to avoid using "return expression" (see AI-318-2).
- Revised and moved the wording of the 4.8(7/2) checks to be consistent with the AI-373 changes.

It's not clear what the rule of 3.10.2(16.1) applies to. Add "For determining {in the above rules} whether ...". Tucker suggests the simpler: "In the above rules, the operand...", as the rest is repetitive anyway.

Pascal is worried that we didn't define statically deeper relationships for many of these new accessibility rules.

3.10.2(13.1/2) is wrong; AI-254 needs to be fixed (see above).

But that's not Pascal's problem. He's afraid that an access discriminant would need a dynamic accessibility level stored in it.

Tucker will take this off-line with Pascal. He then says that you have to make these checks; but there is no object at the point of the checks (so the accessibility wouldn't need to be stored).

AI-418/03 Vector norm

Randy says that he corrected the accuracy requirements to be like the ones preceding them, and defined "g" in the real version (the complex one had "g" defined).

AI-419/04 Limitedness of derived types

We needed wording for interfaces like that for tasks and protected types to cause inheriting of subprograms from progenitors.

3.9.1(3/2) didn't cover progenitors. We need to refer to 7.5 as for why a type is limited.

The wording of 3.7(10/2) includes derived types with the reserved word limited, but that would be different and not intended.

3.9.1(3/2) seems circular. Randy complains that limited derived types are not covered properly. But the existence of the reserved word **limited** doesn't matter; it can't make anything nonlimited. The type would be illegal if one of the progenitors or the parent is nonlimited. So change the wording to "...if the parent type or any progenitor is nonlimited...".

AI-423/06 Renaming, null exclusion and formal objects

Randy explains that a number of changes were made:

- The wording in this AI did not use the technical term "subtype that excludes null".
- The syntax for object renames had null_exclusion added, to match every other kind of declaration.

- The lead-in to the null_exclusion matching rules was corrected to allow the null_exclusion to come with directly from the renames or from an access_definition (only the second case was mentioned previously).
- The first bullets of all of the rules need wording to apply to occurrences in a child unit of the generic whose formal we are talking about. The wording for this was lifted from 3.10.2(32/2).
- The second bullets of all of the rules need the infamous boilerplate to be rechecked in the private part of the instance. (Otherwise, you could have an object with a null_exclusion that has the value null.)
- A number of editorial changes were made.

No one has any comments on these changes, but 12.4(5/2) should be changed in the same way as 8.5.1(3/2) was for AI-409.

AI-424/02 List of language-defined units

Missing units Ada.Wide_Characters and Ada.Wide_Wide_Characters were added.

AI-426/03 Language-defined routines returning abnormal and invalid values

Randy had rewritten the new note 13.9.2(13/2) to be less specific to Unchecked_Conversion.

Drop “if there is a possibility that the result might be invalid” from this note. “can create” should be “can return”.

AI-427/04 Default parameters and Calendar operations

Added large sections of missing wording (mostly descriptions for newly added routines).

Added a default of 0.0 to the smallest parameter of Time_Of, to match the one in Calendar.

Tucker comments that in 9.6.1(70) and others. “through” should be “in”.

Conflicts

[Conflicts are wording changes that occur only because of the intersection of two or more AIs - Editor]

The AI-246/AI-363 combination bullet was split into two (now 4.6(24.8/2) and 4.6(24.9/2)) to make it understandable.

Detailed Review of Amendment AIs

AI-433/02 Examples in the Standard

Erhard notes that 3.3.2(10) is missing a semicolon.

Erhard asks that the lead-in 3.9.4(16/2) sentence say “a limited interface” and “a synchronized interface”.

Ed suggests type “Locking_Queue”. But we use that name elsewhere, and we want its meaning to be obvious. So no change is made.

In 3.9.4(19/2), the parameter types should be Synchronized_Queue instead of Queue.

In 3.9.4(21/2), Jean-Pierre says that parameter colons should be aligned. Alan sees that Person_Name_Name should be Person_Name.

In 3.9.4(22/2), Erhard notes that the penultimate sentence does not need the last comma.

Pascal would like to change the parameter name "Element" to "Person" in all of these routines. The local variable should be changed similarly. This suggestion gets general agreement.

In 3.9.4(23/2), Erhard asks that the first comma be removed.

In 3.9.4(24/2), Tucker suggests "use of the interface".

In 3.9.4(25/2), the space before the "(" is wrong, should be removed. There is a similar space in the 8.3.1 examples.

In 3.9.4(29/2), Tucker doesn't like "can be added when a new type is derived from another". Change that to "used as a progenitor when a type is derived"

In 3.9.4(32/2), Erhard notes that "using" should be "by".

Jean-Pierre suggests removing extra spaces from Wife and Husband in 3.10.1(22): Pascal says we're not changing stuff that isn't broken and not otherwise changed. So no change will be made.

Erhard suggests that 4.3.3(43/2) be changed as follows: "of {an} array... choice[,]{ and} with {an} applicable...".

The wording in 8.3.1(15/2) ("a spelling error") is considered confusing. Erhard suggests "A misspelling in one of these subprograms will be detected by the implementation. Conversely, the declaration...".

The 9.11 example should not consume or produce people. The example is not about cannibalism! So change 9.11(3/2) to say "simulate arrival of the next customer", and 9.11(6/2) to "simulate serving a customer".

In 9.11(7/2), Tucker would prefer "list" than "pool". Actually, "array" would be better. "People" should be "person names". "the space of the next" should "the index of the next" (two places).

In 9.11(7.1/2), remove the second comma. "is promising" should be "promises".

In 9.11(9.1/2) and 9.11(11/2), the assignment of the element should line up with the assignment below it.

For 10.1.2(24/2), Pascal asks if we have a unit named Office. No, we need one here. Add:

```
package Office is
end Office;
```

Tucker would like to put it in 10.1.1. "A unit that is intended to be the root of a subsystem." Then we need "(see 10.1.1) in 10.1.2. Tucker gives up, put it in 10.1.2.

We also need a definition of Office.Locations. Something like:

```
package Office.Locations is
   type Location is ...
end Office.Locations;
```

Pascal suggests:

```
with Ada.Strings.Unbounded;
package Office.Locations is
   type Location is new Ada.Strings.Unbounded.Unbounded_String;
end Office.Locations;
```

Tucker thinks this is overkill, but he loses. We'll use Pascal's version.

11.4.3 and 12.5.5 are out of order in the AI.

Remove with and use clause from 11.4.3(2/2), they're not needed anymore. (And they never should have been on the specification in the first place.)

In 12.5.5(6/2), **access** should be **in out** because entries can't have **access** parameters. (How many people remember that??)

Change Work_Item to a tagged private type. Tucker doesn't like that, so declare a regular type before the generic:

```
type Root_Work_Item is tagged private;
```

Erhard asks that an "a" is added 12.7(12/2) to: "{a} generic package"

Erhard asks that 12.7(17/2) be replaced by "Example of an instantiation of a package with formal packages:"

In 12.7(13/2) Generic_Mapping isn't defined. We should define that; but we already have such a package defined in the containers library. So we'll use Ada.Containers.Ordered_Maps here. Change Generic_Join to Ordered_Join.

Types String_Id and Symbol_Info are also not defined. Make the second part of the example into a package, and define the missing types as "...". (It would be nice if they could be private, but we didn't adopt some form of AI-359.) Add a with_clause for Ada.Containers.Ordered_Maps to the package.

Approve AI with changes: 12-0-0.

Detailed Review of Regular AIs

AI-415/03 Presentation changes to the Standard

Tucker objects to removing the italics from "of". But "of" is not a technical term! Tucker thinks it is. What will SC22 say when we announce that we're defining "of"? We could put the whole phrases into italics (defining the phrases), but then the phrases should be in the index (they're not currently, nor is "of"). Tucker prefers that.

Jean-Pierre asks if the mathematical terms are normative. No, they're described under "definitions" (which otherwise doesn't have any definitions).

Item 2 in discussion: "argubly" should be "arguably" (but this is going to change anyway to reflect Tucker's suggestion).

Get rid of extra commas from 3 and 4 of the discussion.

Approve AI with changes: 12-0-0.