Minutes of the 43rd ARG Meeting

18-20 February 2011

St. Pete Beach, Florida, USA

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins (as a proxy for John Barnes), Gary Dismukes (except last two hours of Sunday), Bob Duff, Erhard Ploedereder, Ed Schonberg, Tucker Taft.

Observers: Greg Gicca.

Meeting Summary

The meeting convened on 18 February 2011 at 9:20 hours and adjourned at 16:00 hours on 20 February 2011. The meeting was held in the Sabal / Canary meeting room at the Sirata Beach Resort. The meeting covered the entire agenda.

AI Summary

The following AIs were approved :

AI05-0111-3/07 Subpools, allocators, and iterators (8-0-0)
AI05-0176-1/10 Quantified expressions (8-0-0)
AI05-0202-1/03 Task_Termination and Exceptions raised during finalization (7-0-1)
AI05-0224-1/02 No_Task_Allocators and classwide returns (7-0-0)
AI05-0232-1/02 Hole in AI05-0067-1 (6-0-2)
AI05-0238-1/01 Split behavior at the limit (7-0-1)
AI05-0239-1/01 Hyphens in mode conformant (7-0-1)
AI05-0242-1/01 No_Implementation_Units restriction (8-0-0)

The following AIs were approved with editorial changes:

AI05-0092-1/16 More presentation issues in the Standard (8-0-0)
AI05-0119-1/01 Package Calendar, Daylight Savings Time, and UTC_Offset (7-0-1)
AI05-0131-1/02 Class-wide operations for formal subprograms revisited (7-0-1)
AI05-0153-3/05 Subtype predicates (7-0-1)
AI05-0167-1/06 Managing affinities for programs executing on multiprocessors (7-0-1)
AI05-0182-1/02 Preciseness of S'Value (5-0-3)
AI05-0188-1/11 Case expressions (5-0-3)
AI05-0192-1/01 Behavior of 'Input for types with constrained first subtype (8-0-0)
AI05-0194-1/02 The default value of S'Stream_Size (7-0-0)
AI05-0198-1/01 Inheriting abstract operations for untagged types (6-0-2)
AI05-0201-1/01 Independence and components of atomic objects (7-0-0)
AI05-0214-1/04 Default discriminants for limited tagged types (6-0-2)
AI05-0220-1/01 Definition of "needed component" (8-0-0)
AI05-0221-1/01 Correction to "preelaborable initialization" (8-0-0)
AI05-0225-1/04 Call using constant protected objects (7-0-1)
AI05-0227-1/03 Identifier equivalence (6-0-2)
AI05-0228-1/03 Default initial values for scalar and array types (7-0-0)
AI05-0235-1/03 Accessibility of explicitly aliased parameters (5-0-3)
AI05-0237-1/01 Ancestor types for synchronized formal types (7-0-1)
AI05-0240-1/01 Rewordings from First Editorial Review (8-0-0)
AI05-0241-1/02 Aspect-related restrictions (7-0-1)
AI05-0244-1/01 Ancestor parts that are qualified or parenthesized (6-0-2)

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

AI05-0110-1/01 Characteristics of generic formal derived type are not inherited (7-0-0)
AI05-0139-2/08 Syntactic sugar for accessors, containers, and iterators (6-0-2)
AI05-0183-1/08 Aspect specifications (7-0-1)
AI05-0190-1/09 Global storage pool controls (8-0-0)
AI05-0197-1/01 Dispatching when there are multiple inherited subprograms (7-0-1)
AI05-0200-1/01 Mismatches in formal package declarations (4-0-3)
AI05-0212-1/05 Accessors and Iterators for Ada.Containers (7-0-1)
AI05-0229-1/02 Specifiable aspects (5-2-1)
AI05-0234-1/03 Hole in AI05-0051-1 (6-0-2)
AI05-0236-1.01 Additional problem with preelaborated generics (7-0-1)
AI05-0243-1/02 Clarification of categorization (7-0-1)

The following AIs were discussed and assigned to an editor:

AI05-0115-1/05 Aggregates with components that are not visible
AI05-0125-1/03 Nonoverridable operations of an ancestor
AI05-0206-1/02 Remote_Types and Remote_Call_Interface packages should be able to depend on Preelaborated packages
AI05-0218-1/01 Generics and volatility

Detailed Minutes

Previous Meeting Minutes

Approve minutes by acclamation.

Date and Venue of the Next Meeting

Our next meeting will be in conjunction with Ada Europe in Edinburgh, Scotland June 24-26, 2011.

We'll have a phone meeting fairly soon to vote on completions of the AIs still open after this meeting. That will be on. March 17th, 2011 at 1pm EST (-5 UTC). This will be a “real” meeting, with binding votes.

RM Review Plan

Randy explains the plan. There will be a second ARG review; everyone will be assigned a section that will be different from the one they reviewed last time. This second review will start as soon as the draft is ready (probably a couple of weeks after the phone call). That will be followed by a National Body Review (probably about 8 weeks), starting sometime in May. After that, we'll do a formal submission (probably via ANSI) and the draft will go through the formal approval steps (WG 9, SC 22, JTC 1).

AARM Simplification

Randy notes that Bob had asked him to consider removing the in-line index entries and glossary entries from the AARM. He notes that he has had several complaints about the index entries. He also notes that there is a chance that removing other entries would change AARM paragraph numbers (which we generally avoid changing).

There are other similar things, such as the entries for Annex M items, but those tend to not be obvious (especially for implementation-defined things where they are often extracted from much longer paragraphs), so he is inclined to leave them.

Bob says he meant to eliminate those things, and never did. Tucker notes that the in-line things hurt the readability the most. Randy will remove the index entries and glossary entries from the online and PDF AARM (and not remove any of the other things).

ASIS

Jean-Pierre Rosen sent out ASIS assignments. We'll set a deadline for these some time after the end of the National Body Review – Ada 2012 is the number one priority at this point.

Thanks

Thanks to Greg Gicca for the fine arrangements and the beautiful weather during the meeting.

Thanks to Adam Beneschan for providing problems whose complexity is in inverse proportion to their importance.

Old Action Items

Pascal didn't do the item that has been open for a long time. This was something he wanted done, and there is nothing critical about this, so we will just remove it from the Action Item list.

Ed did not update AI05-0215-1.

Gary didn't send wording for AI05-0115-1 because it wasn't clear that a consensus on how to approach the problem had been reached.

Two other items were late and do not appear on the website as part of the 'official' AIs for the meeting. The AI05-0212-1 example was sent after the deadline Wednesday, as was the discussion section for AI05-0234-1.

New Action Items

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

Everyone:

- Complete their ASIS assignments as previously sent by Jean-Pierre Rosen.

Steve Baird:

- AI05-0110-1
- AI05-0190-1 (help Bob Duff)
- AI05-0197-1 (with help from Tucker Taft)
- AI05-0218-1
- AI05-0229-1 (help Randy Brukardt)
- AI05-0234-1
- Check whether there is a problem with explicitly aliased parameters of an entry. (See discussion of AI05-0235-1.)

Randy Brukardt:

- AI05-0212-1
- AI05-0229-1 (with help from Tucker Taft, Steve Baird, and Bob Duff)
- AI05-0241-1
- AI05-0243-1

Editorial changes only:

- AI05-0092-1
- AI05-0119-1
- AI05-0131-1
- AI05-0153-3
- AI05-0182-1

- AI05-0188-1
- AI05-0192-1
- AI05-0194-1
- AI05-0198-1
- AI05-0201-1
- AI05-0214-1
- AI05-0220-1
- AI05-0221-1
- AI05-0225-1
- AI05-0227-1
- AI05-0228-1
- AI05-0235-1
- AI05-0237-1
- AI05-0240-1
- AI05-0241-1
- AI05-0244-1

Bob Duff

- AI05-0167-1
- AI05-0190-1 (with help from Steve Baird)
- AI05-0229-1 (help Randy Brukardt: create contents of Annex K.2)
- AI05-0236-1

Brad Moore:

- AI05-0206-1

Jean-Pierre Rosen:

- Create an !ASIS section for AI05-0188-1 (see October minutes)

Ed Schonberg:

- AI05-0200-1
- AI05-0215-1 (see October minutes)

Tucker Taft:

- AI05-0115-1
- AI05-0139-2
- AI05-0183-1
- AI05-0197-1 (help Steve Baird)
- AI05-0229-1 (help Randy Brukardt)
- Create an AI to define No_Implementation_Identifiers and more – see discussion of AI05-0242-1.

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs) and Ada 2005 AIs (no SIs were considered at this meeting). The AIs and SIs are presented in numeric order, which is not necessarily the order

in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

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

Detailed Review of Ada 2005 AIs

AI05-0092-1/16 More presentation issues in the Standard

Correct the Summary: “time-type” should be “time type”.

Steve notes that 9.6(28) has “(see D.8)”, there doesn't seem to be any reason for this cross-reference. Remove it, and add that to this AI.

Approve AI with changes: 8-0-0.

AI05-0110-1/01 Characteristics of generic formal derived type are not inherited

Delete the “are not inherited” from the subject, start with “Inheritance of”.

The summary: The characteristics of generic formal derived types are inherited (<current summary>).

The change for 12.5.1(21/2) does not work because it doesn't include subprograms. The AARM note is wrong, operations are not characteristics. Steve complains that 3.4 seems to imply that primitive subprograms are characteristics. He's right, but that seems wrong – primitive subprograms have their own inheritance and overriding rules, we surely don't want to mess with those.

Randy suggests just adding “characteristics and operations”. Bob suggests “and inherited primitive operations”. Tucker suggests “and primitive subprograms”.

Tucker says that this doesn't say what we're deriving from. It needs to say that it is inheriting the characteristics and primitive subprograms from the ancestor type.

Tucker would like to make the wording parallel to 7.3(16) [private extensions].

Steve wonders if 7.3(16) should say “characteristics” rather than “components”. It surely has to in 12.5.1.

Approve intent: 7-0-0

AI05-0111-3/07 Subpools, allocators, and iterators

The changes mostly had to do with task termination.

Tucker wonders if this will be implemented soon. It's not clear, but Bob is probably interested in doing so in GNAT.

Approve AI: 8-0-0.

AI05-0115-1/05 Aggregates with components that are not visible

Gary explains how we are still stuck on these cases.

In the question, F1 should be C1.

Tucker explains his wording.

Bob suggests swapping the two sentences, since the second one is more important.

Gary notes that if we are redefining “descendant”, it is uncomfortable to do it here.

It is suggested that this entire rule is redundant, as it follows from the other rules for visibility. Maybe a large AARM note is enough. But this clearly is very confusing at best and ambiguous at worst.

Ed notes that GNAT allows all of these examples (including using the clearly invisible component).

Tucker says that perhaps we should say that the view of your parent is a characteristic that comes through inheritance like every other characteristic.

The views of the ancestors that you see (are descended from) are a characteristic that you inherit through your parent.

Ed says that this is usually a dynamic notion. Randy notes that this is usually used in legality rule contexts, so views always come into it. It is the dynamic uses that probably are already wrong.

Randy is still worried that there are other problems. We need to look at all uses of the term “descendant” (he found 54 clauses that use the term).

It is the *current* view of the parent type that matters, not the one at the point of derivation. Specifically, the view that a type has of an ancestor is the view that the current view of the parent type has of that ancestor.

It would be best to change the wording for the extension aggregate case to use “descendant” rather than "derived from" somehow in order to remove any doubt.

Tucker will take another shot at this one.

Tucker notes that the formal derived type matching has a similar problem. Do you even know that you are descended from some particular ancestor? That could also depend on visibility.

Steve writes up an example:

```
procedure Formal_Derived is

   type T1 is tagged null record;

   type T2 is new T1 with null record;

   package Pkg is
      type T3 is new T1 with private;
   private
      type T3 is new T2 with null record;
   end Pkg;

   generic
      type T is new T2 with private;
   package G is
   end G;

   type T4 is new Pkg.T3 with null record;

   package body Pkg is
      package I is new G (T => T4); -- legal??
   end Pkg;

begin
   null;
end Formal_Derived;
```

The instance marked "legal??" should also be illegal.

Just keep this alive.

AI05-0119-1/01 Package Calendar, Daylight Savings Time, and UTC_Offset

9.6.1(42/2):

“Returns, as a number of minutes, the result of subtracting the implementation-defined time zone ...”

Delete the first sentence of the AARM note.

Approve AI with changes: 7-0-1.

AI05-0125-1/03 Nonoverridable operations of an ancestor

The wording is supposed be an outer bullet, not an inner bullet.

Gary and Steve try to explain why this wording doesn't work.

The last e-mail is from Randy, not Gary.

When there is an interface, you are getting a new slot with this wording. That seems like a problem.

Tucker points out that this example is currently legal, so if this required a different slot, we would have an inconsistency.

Erhard notes that this is maintenance hazard. It's a sort of a preference rule.

That is no good. We could make this example illegal, which is incompatible.

Randy wonders how you would fix this incompatibility if it happened. Tucker thinks that adding the interface to grandparent would make it legal. That's not going to going to work well if the grandparent is out of your control; but that should be fairly rare when this would happen (which requires being within the hierarchy of units of the grandparent).

Tucker will make an attempt to fix this wording.

AI05-0131-1/02 Class-wide operations for formal subprograms revisited

Steve notes that in the !discussion. “This is the only part of the wording changes that is motivated by this "homograph collision" problem.” This is not quite true, because of the wording change 8.4.

The wording does not apply to constructor functions (those without a controlling parameter). Nothing happens for such subprograms.

Second example has typos: “end Pkg{1};” “procedure Foo (X1 [;]{:} T1)

Approve AI with changes: 7-0-1.

AI05-0139-2/08 Syntactic sugar for accessors, containers, and iterators

Steve asks what happens if you have defined the Indexing aspect on a private type, and the full type is an array type. What happens in the body when you do indexing?

```
package P is

   type Priv is private
      with Indexing => Idx;
```

```
      function Idx (A : Integer) return Foo;

private
   type Priv is array (Integer range 1 .. 10) of Foo;
end P;

package body P is
   Obj : Priv;

   Obj(B) := ...; - ???
```

Steve then asks about a default expression using indexing in the visible part, and then the same expression is used in the body. Does this expression conform??

Tucker thinks that the predefined operation has to hide the specified one. Otherwise it gets too complex.

Bob thinks this is the same as:

```
package P is
   type Priv is private;
   function "+" (A, B : Priv) return Priv;
private
   type Priv is Integer;
end P;
```

"+" does not reappear here. Bob thinks that indexing should work the same way.

Randy wonders if it shouldn't be illegal, because we are trying to (implicitly) redefine an aspect. Generally, we only allow a single definition of an aspect for each type.

Tucker agrees with Bob, user-defined hides predefined.

Tucker wonders if there is a similar issue for implicit dereference. But that has to be a type with an access discriminant, so it can't be an access value. So there is no problem.

Ed asks what the purpose of aspect Iterator_Element is, Tucker explains that it tells the compiler what the element type is for the iterator "of" syntax.

These are operational aspects (otherwise they couldn't be specified on partial views), but that causes issues.

What happens for inheritance of these? 13.1(15.1/2) says, but that doesn't work for these as it doesn't specify anything for tagged types. The inheritance of partial and full views is tricky; if they do not have the same "value" for the aspect, it has to be specified or the type is illegal.

We should change 13.1(9.1/1) from "of the type" to "of the entity"; we want operational aspects on more than just types. 13.1(8.1) also seems to assume that all operational aspects are on types. Tucker will look at this as part of AI05-0183-1.

Approve intent of AI: 6-0-2.

AI05-0153-3/05 Subtype predicates

Typo: "contraint check".

Steve finds the second bullet of the Legality Rules for "predicate static" to be odd. It's a bit weird, but Bob explains that it matches how it works, and it allows unification of case expressions and memberships.

Steve still dislikes "if it were replaced". Ed agrees.

Randy says that Bob had previous wording for the membership bullet that worked fine, Bob is unsure.

- a membership test whose simple_expression is the current instance, and whose membership_choice_list meets the requirements for a static membership test (see 4.9);

- a case_expression whose *selecting*_expression is the current instance, and whose *dependent*_expressions are static expressions;

Approve AI with changes: 7-0-1.

AI05-0167-1/06 Managing affinities for programs executing on multiprocessors

Should Dispatching_Domain be an aspect? Yes. Should we have a pragma? Yes, although will be born obsolescent.

What is obsolescent is that some pragmas can be used to specify aspects, this is just one of those.

Bob will redo this to describe this as an aspect.

Randy notes that using the pragmas to specify properties of the main subprogram is more convenient than aspects, so we will still encourage that. There is agreement with the point. (This applies to AI05-0229-1.)

Approve AI with changes: 7-0-1.

AI05-0176-1/10 Quantified expressions

Tucker says this looks a lot like an aggregate, he would like an iterator aggregate. We tell him that it is way too late for that; just because the syntax is similar does not give us license to invent new unrelated stuff.

Bob wonders why there a vertical bar in the !proposal text. That is referring to mathematical notation.

Approve AI with no changes: 8-0-0.

AI05-0182-1/02 Preciseness of S'Value

Change wording to:

An implementation may extend the Wide_Wide_Value, Wide_Value, and Value attributes of a character type to accept strings starting with “Hex_” (ignoring case) for graphics characters and those with a code position smaller than 16#100#, and three character strings of the form “'*nongraphic character*'”.

Approve AI with changes: 5-0-3

AI05-0183-1/08 Aspect specifications

!summary “types and object{s}”.

Erhard wonders if we are considering these preferred to the older ways of specifying aspects. He suggests that we might consider the other version as obsolescent. He is concerned about having multiple ways to do the same things.

Tucker would argue to get rid of pragmas on subprograms and types. Randy says he was thinking the same way. Attribute_Definition_Clauses can do a few things that aspect clauses can't (hiding specifications in the private part, for instance), but that isn't true for the pragmas. And the pragmas have the overloading problems for subprograms.

Incorporate improved wording for operational aspects and inheritance of aspects (see discussion of AI05-0139-2).

Approve intent: 7-0-1.

AI05-0188-1/10 Case expressions

The current rule is to recheck (on variant parts). That follows from the standard handling of Legality Rules in generic instances. Randy notes that this affects clients, they may not be able to create the objects that they need to use the generic.

Tucker would prefer that we don't add any test cases for the variant case. There is some discomfort with that.

We will just make an exception for case expressions, and Steve will minimize this wording.

Several people object to this resolution. A straw poll is requested.

Make variants consistent and case expressions consistent (no recheck on either): 5-1-1.

The argument for not checking variants is that it isn't much worse than things you can already do. For example:

```
generic
   Low : in Integer;
package G is

   subtype Foo is Integer range Low .. Formal'Last;

   type Rec (D : Foo) is record
      case D is
         when -10..-1 =>
            Foo : Float;
         when others =>
            null;
      end case;
   end record;

end G;
```

Approve intent: 6-0-2.

AI05-0188-1/11 Case expressions

Steve had sent new wording on Friday afternoon.

Ed wonders about “instance of a generic unit”, he doesn't know of any other instances: It is not an “instance of a caesar salad”. Can't we just say “instance”? Bob says that we inherited this from Ada 83 and rarely say “instance” by itself.

"which" should be “that” in the 4.5.7 wording.

Change the change to “except within an instance of a generic unit” in 4.5.7.

and “If the discriminant is of a static constrained scalar subtype, then, except within an instance of a generic unit, each non-**others** discrete_choice...” for the 3.8.1 change.

Drop the existing comma before the then.

Ed suggests instead:

“If the discriminant is of a static constrained scalar subtype, then, if not within an instance of a generic unit, each non-**others** discrete_choice...”

No help, we'll use the previous version.

Reopen the AI, add this wording change, then approve the AI with changes: 5-0-3.

AI05-0190-1/08 Global storage pool controls

Looking at the finalization wording:

For an allocator of an anonymous access type that {is declared in a component_declaration} [declares a component], the freezing point of the composite type that declares the component.

Tucker suggests that the master of the anonymous access type is sufficient. Randy objects, the types belong to the subprogram spec, not some call. Tucker and Bob claim that 6.4.1(9) + 3.10(16/2) + 3.10.2(13/2) mean that the type is per-call.

So we don't need "collection master" at all; it is just the "master of the access type."

"Collection point" could be called "Initialization point".

Cases missing there: discriminants that are not coextensions, and anonymous function results.

Bob will to try simplify this wording off-line.

Erhard is confused by the note – he's pretty sure that allocators are not exceptions. Change the note to end with "...to ensure that all allocators use the default pool."

Make the second restriction more consistent with the existing restrictions.

"Allocators are not permitted as the actual parameter to an access parameter. See 6.1."

Approve intent: 8-0-0.

AI05-0190-1/09 Global storage pool controls

Bob sent a rewording of the finalization rules.

First sentence of second paragraph: Each nonderived access type {T} has an associated *collection***,** which is which is the set of objects created by allocators of {T}[the type], or of types derived from {T}[the type].

"...removes {an}[the] object..."

Remove the coextension sentence, this is defined by 7.6.1(9.1/2).

"For an allocator for an access parameter, the call that contains the allocators."

"For the type of an anonymous allocator defining the value of an access parameter, the call that contains the allocator." ("anonymous allocator").

3.10.2(14/3) has two periods at the end of the first sentence in the draft RM. (One should have been deleted.)

Collections are objects, so that these are defined in the proper order for finalization (which reverses the order).

The "otherwise" case covers the access result case, that doesn't work. We need a separate bullet for that.

Tucker does not think that anonymous objects related to calls are being "declared".

Tucker says that the order of finalization is only significant if the master includes a list of declarations. Otherwise, the order really isn't significant. That probably was the intent of his rewrite in AI05-0051-1, but he went way, way, too far. [Editor's note: I don't think this is worth complicating the wording; since the parameters can be evaluated in an unspecified order, so can the finalizations occur in an unspecified order. If it is easy to do, though, go ahead.]

We try to find counter-examples, but no one has any. We still need some sort of special rule for access results.

Bob and Steve will take another attempt at this wording.

AI05-0192-1/01 Behavior of 'Input for types with constrained first subtype

Tucker notes that the question should say that is the “result {subtype}”.

Steve notes that we don't want explicit specification of the attribute to be different than the inherited version.

Randy wonders if that would be a problem if the original attribute (T1'Input in the question) was explicitly specified. That would be different than the derived type.

We probably want inheritance this way; that argues that 13.13.2(50/2) is wrong. This is the same as “&” and other derived operations; you might inherit something that you cannot write. We want it to apply only to non-inherited operations, add wording to that effect (and change the AI to a binding interpretation).

Steve asks about changing the number of discriminants on a stream attributes (or adding defaults).

13.13.2(25/2) is only relevant if the parent operation has been specified and visible; otherwise the attribute is created from scratch. Add an AARM ramifications note to say that we never inherit from default implementations.

Tucker says that it would be better to say for “default” stream attributes. Gary notes that the term is “default implementations”. So the 13.13.2(50/2) wording should say, “... for {the default implementation of} the stream-oriented attributes...”

Add an AARM note to say that inherited operations may have a different profile.

Randy will rewrite this again.

Steve does not like the wording of 13.13.2(25/2). He says that the wording seems to say that this applies for attributes.

Tucker adds “{specified and} available” to 13.13.2(8.1/2) and 13.13.2(25/2).

Answer the question (Yes.). Add to the discussion the example where T1 has a specified attribute; in that case, the answer is (No.).

Erhard doesn't like this result. We point out this is the same behavior as for any function inherited from T2; we can't change that even if we wanted to.

Approve AI with changes: 8-0-0.

AI05-0194-1/02 The default value of S'Stream_Size

Summary, second sentence: “does {not} change”.

Approve AI with change: 7-0-0.

AI05-0197-1/01 Dispatching when there are multiple inherited subprograms

...otherwise{ (if the others are null)} the action is that of an arbitrary one of the operations.

Tucker does not like the wording (but the reason was not recorded - sorry).

Small bug in the example: `X : {Pack2.}Pack3.T3;`

Steve and Tucker will try to improve this wording.

Approve intent: 6-0-1.

AI05-0198-1/01 Inheriting abstract operations for untagged types

Tucker notes that the predefined routine has parameters “Left” and “Right”. So the full conformance fails and they just are hidden from all visibility. This again has the right effect.

Approve AI with changes: 6-0-2.

AI05-0200-1/01 Mismatches in formal package declarations

The summary is confusing. This AI adds rules so that checking on a formal instance is the same as that on a “regular” instance.

“derived from T” is weird. We want it to be an actual type that would match T, but different from any other type in the program.

Ed will try to reword the second paragraph.

Tucker hopes that there is no work if the entire thing is given by <>. That is intended.

Tucker is wondering if there is a rule that requires using <> for things that depend on other parameters.

```
generic
   type T is ...
   type AT is array (Integer) of T;
package Gen is ...

generic
   type Foo is ...
   package G is new Gen (T => <>, AT => Foo); -- ??
```

Ugh. This also should be illegal.

The issue is formal parameters that depend on other formal parameters; if the original is <>, then other parameters that depend on it also need to be <>.

Approve intent: 4-0-3

AI05-0202-1/02 Task_Termination and Exceptions raised during finalization

Tucker wonders if he covered the case where an exception is propagated, and then the finalization of the task also propagates an exception.

Steve notes that for exceptions propagated by finalization, you propagate Program_Error, not the original exception.

What about a finalization that fails when the task is aborted? If Program_Error is raised, it will disappear in that case, we keep abort.

Erhard notes that the Exception_Occurrence could still be set to Program_Error if the finalization fails, even if the task was aborted.

That seems like a good idea; the Exception_Occurrence should have the Program_Error.

Tucker will reword this paragraph, and send it to us later.

Steve worries that an implementation that uses exceptions to implement abort will have to do work. Tucker notes that that only requires a bit to note whether or not the finalization failed.

If you are monitoring tasks you want to know about failures, especially these that can't be detected any other way. Otherwise there might be storage leaks, missing device resets, etc.

So we do want to do this.

Tucker finishes the new wording while we discuss this; but we decide to consider it later once everyone has had a chance to read it.

Gary notes a typo in paragraph 2 of the !discussion, “anthing” should be “anything”.

1st paragraph of the question: “specified” is misspelled.

Approve intent: 6-0-2.

AI05-0201-1/01 Independence and components of atomic objects

The summary is missing.

The change in 9.10(1/3) isn't marked, but it is just adding “Atomic” to the list.

Tucker suggests changing the AARM note to "Note that the components of an atomic object are not necessarily atomic."

The question is weird, but that was the question that was asked. The answer is not helpful, however.

So change the answer to “(No, but it can effect whether the components are independently addressable)”.

A question arises whether Atomic should be allowed to change the representation of a type to make it Atomic. We eventually decide that this does not matter here. (It falls into the category of a question not asked, which is it best not to try to answer with our limited time.)

Tucker thinks that **aliased** should imply independent-addressability. Steve mutters something about **aliased** or **tagged**. This seems like another question not asked.

Approve AI with changes: 7-0-0.

AI05-0202-1/03 Task_Termination and Exceptions raised during finalization

On Sunday, we consider the updated AI that Tucker e-mailed us. No problems are noted.

Approve AI: 7-0-1.

AI05-0206-1/02 Remote_Types and Remote_Call_Interface packages should be able to depend on Preelaborated packages

Steve wonders if you can have a child package on a different partition that can see the things in the private part. This seems like a problem that would exist for Remote_Types anyway, since it does not have any restrictions that apply in the private part.

We can't decide on this one, we would like to hear from Brad. So we table this one for now.

AI05-0212-1/05 Accessors and Iterators for Ada.Containers

Constant_Reference_Type and Reference_Type should just be **private**, not **with private**.

Typo: “convinient” => “convenient”. Bob says that there are 14 AIs that have that spelling, should fix them all. [Editor's note: I only found 5 such AIs; most of the occurrences are in e-mail; we don't fix errors in e-mail.]

Randy asks about the change from Pure to Preelaborate. Tucker wonders if we could just change Controlled to Pure (also dropping Remote_Types). No one can think of a real problem with that. There is “state” involved, but that state

is managed solely by the implementation (not by the user), and there is no way to depend upon it. (A Pure Finalize routine could not depend on global variables, which might be harder to write, but tough.)

Add that to this AI, and then leave the bounded packages as Pure.

For Sets, there cannot be a Reference routine (there is no Update_Element) in the outer generic, because that requires moving the element.

The inner one has one based on Update_Element_Preserving_Key.

There is some discussion about a derived set type for which Indexing is defined (inside the inner generic). Randy thinks that would not work practically.

Tucker will look into how bad that would look/work. He will get that done soon.

The example looks good, do need to compile and run it. There is a problem with the first iterators; the cursor can't be used with other objects (these are all different types). Tucker does not like that Next is not initialized. Ed says that is OK, as the vector can't be empty. Maybe a precondition is needed. Ed will continue to work on this.

Approve intent: 7-0-1.

AI05-0214-1/04 Default discriminants for limited tagged types

Steve asks one of his famous questions. What happens for:

```
package P is
   type Priv (A : Natural) is private;
private
   type Priv (A : Natural) is tagged ...
end P;

with P;
package Q is
   type P2 (B : Integer := 10) is new P.Priv(B);
end Q;
```

P2 would now have 'Constrained = True (the attribute is calculated at run-time), while in Ada 95 this is actually assignable.

Tucker claims that this type is not a tagged type, even at run-time. So 'Constrained is in fact computed in the normal way. (The wrapper model applies here.)

Perhaps add an AARM note to explain that if the type is derived from a tagged type but is not a tagged type itself (see above for an example), the attribute follows the rules for an untagged type.

Approve AI with changes: 6-0-2.

AI05-0218-1/01 Generics and volatility

Add an answer “(Yes.)”

!wording is missing after (See summary.)

The discussion should mention that this incompatible; some actual types are no longer legal for certain formal types.

Tucker wonders why this only applies to formal derived. The reason that the rule didn't apply before is that all of the formal types other than derived already are pass-by-copy.

Tucker now thinks that the problem is related to the (aliased) components. Only formal derived and formal arrays have components that could be a problem. (There aren't any aliased discriminants.) Volatile array components would have the same problem.

It would seem that we need a similar rule for formal arrays.

Also note that tagged components would have the same problem.

Steve previously noticed a problem with view conversions of arrays (that is, in an in out parameter) that prevents volatile components.

Explicitly aliased parameters seem like an issue here (if they have a non-volatile formal type and the actual is a volatile type). We never want to have a reference to a volatile object for which the code does not know that the object is volatile. (By-copy is OK, by-reference never is.)

Steve will take this one back and consider other cases and other problems.

Keep alive: 7-0-0.

AI05-0220-1/01 Definition of "needed component"

Typo: A line in the !discussion starts with !question, which screws up the formatter.

Gary would like a space in “runtime” in the question. OK.

Approve AI with changes: 8-0-0

AI05-0221-1/01 Correction to "preelaborable initialization"

Does type T2 [has]{have} preelabora[a]ble initialization

Approve AI with changes: 8-0-0.

AI05-0224-1/02 No_Task_Allocators and classwide returns

Approve AI: 7-0-0.

AI05-0225/03 Call using constant protected objects

Steve suggests that the wording should allow only what we want to allow ('Count).

“The name of a protected procedure or entry of a constant view of a protected object is legal only as the prefix of attribute Count.”

Should we allow anything else? Probably not.

“If the target object is a constant view, the name of a protected procedure is illegal; the name of a protected entry shall be used only as the prefix of attribute Count.”

Is target object well-defined? It is for calls, but is it well-defined for renamings?

Tucker would like to fix the definition of target object to include renames, and 'Access.

That might work; Tucker will take this one.

AI05-0225-1/04 Call using constant protected objects

On Sunday, we consider an updated AI that Tucker has e-mailed us. No one notices any problems.

Approve AI: 7-0-1.

After we vote, Tucker announces that there is a bug. The wording needs to say “a protected entry” rather than just “an entry” as it is OK to call a task entry of a constant task. So fix 9.5(7.1/2).

Steve wonders if this would a problem for a synchronized interface (for by-entry). This rule doesn't apply to those.

AI05-0227-1/03 Identifier equivalence

Typo: “so this is [a] matching [to] the defining”

Approve AI with changes: 6-0-2

AI05-0228-1/03 Default initial values for scalar and array types

Tucker notes that we don't want the derived type rule for True to False to apply here if this is a Boolean type. Similarly, we don't want this to allow a default.

```
type My_Bool is new Boolean
   with Default_Value;
```

Ugh. The AI05-0183-1 definitely allows this; we'll need rules to override those.

Steve worries about the freezing. Randy says we don't need to say that, because static expressions always freeze. But that is wrong. We want these to freeze at the point where the associated entity is frozen.

This seems to be an issue for AI05-0183-1 in general. You don't resolve these until the freezing point, so you can't even know if these are static until we reach that point. Tucker is directed to deal with that as well.

In 3.3.1(21): drop “explicit” in the new wording.

We need to make sure that this aspect gets inherited. This is an operational aspect. Tucker's new wording hopefully will handle this.

Tucker is worried about a parameter subtype that cannot represent the default value (or the current value). We are requiring that it preserve the current value without a check. That ain't gonna work. Let's think about that over lunch.

After lunch (and a visit to the Dali museum for most of the group), we return to this AI.

Tucker says that the solution is to make Default_Value a representation aspect. That makes it illegal to specify on a derived type that has inherited subprograms. Add an AARM Note to explain that **out** parameters need to be large enough support 'Base of the type. That has to be true for all inherited routines.

Bob notes that it would be OK to change the Default_Value for a derived type, the problem is adding it to a type that doesn't have one. The rules already make it OK if there aren't any user-defined operations; it isn't worth the headache to go further than that.

Approve AI with changes: 7-0-0.

AI05-0229-1/02 Specifiable aspects

We start by discussing the format and contents of the aspect annex.

Do we want aspects in the aspect annex that can't be specified with aspect_specification? For instance, enumeration rep clause. Yes, but we'll need to add wording to say that you use an enumeration rep clause rather than an aspect_specification.

Randy will send Bob a complete list of aspects.

We discuss the concern about having multiple ways to specify aspects. We prefer that new code uses aspects, but we still need to support the other methods for compatibility. And we don't want to change tons of existing library packages (changing pragma Pure to an aspect, for instance, even if that is better).

Tucker notes that in the past, the way that we've done this is to move things to the Obsolescent Annex (Annex J). Then they are no longer mentioned in the body of the standard and there is less appearance of multiple ways to do things.

He goes on to suggest that we move pragmas on types, subprograms, and objects into the Obsolescent Annex. Pragmas on packages stay, and no changes for attribute_definition_clauses. Aspect_specifications are far superior to pragmas for subprograms, and preferable for most types.

Steve notes that would imply making pragma Pack obsolescent. Tucker says, yes, that's intended. There is concern that there will be push-back from that.

It is pointed out that because restriction No_Obsolescent_Features, moving anything to Annex J is effectively incompatible. We could make that restriction obsolescent (leading to laughs). But seriously, no one has to use that restriction, and it is much less useful than the No_Implementation_whatever ones. So this is unlikely to be a problem in practice.

AttacheHandler => Attach_Handler

Randy asks if the wording is OK.

“can” => “may” (in many places). [Editor's note: some of these are in existing wording!]

Drop the wording about the aspect_mark, it is always true. [Editor's note: for pragmas and attributes the definition of the aspect name is a long ways away in the standard, so I added AARM notes to clarify.]

Should talk about the “type of the aspect”, that will automatically cause a resolution rule.

If “values” aren't static, they get evaluated at the freezing point.

Drop the “with the value True” from all of the pragmas as to what can be specified, they will stay static. AI05-0183-1 will need updates to handle freezing of these expressions properly.

Remove the comma “{,} and shall” in the addition after C.3.1(7/3).

Require the procedure to be parameterless, then we don't need that legality rule.

C.3.1(9): If the Interrupt_Handler aspect of a protected procedure is True, then...

Try avoid “specifies” use “identifies” (as in C.3.1(10)).

Tucker thinks that we want to make Preelaborable_Initialization an aspect. For “normal types”, we would only allow it to be confirming.

Tucker and Steve offer to help with this, they are not quite sure why when thinking about it more carefully. Randy will split up the work.

Bob is still on the hook to write the annex K items.

Approve intent: 5-2-1.

Gary says that he voted against as putting stuff in the Obsolescent Annex rubs him the wrong way (he says it not a strong objection). Jeff Cousins says that he voted against as he is concerned about people maintaining legacy code, and he thinks that John does not like using aspects everywhere.

AI05-0232-1/02 Hole in AI05-0067-1

Approve AI: 6-0-2.

AI05-0234-1/03 Hole in AI05-0051-1

Steve tries to explain the problem and the solution.

In the second paragraph of 4.8(10.3), move the “of the allocator” ahead of the parens.

Tucker would like to unify build-in-place and not build-in-place for coextensions. This is the rule 3.10.2(14.4/3). Steve points out that would involved defining what BIP means for lots of cases not currently possible (copying of a global object, for instance). Limited initialization is limited to particular kinds of expressions.

Randy notes that BIP and non-BIP both use the wording “becomes a coextension”.

Steve and Tucker will take this off-line and try to improve the wording.

On Saturday morning, we revisit this AI.

Tucker explains that he believes that the accessibility checks are OK. Start with 6.5(21/3). Note that this rule starts with “any part”. Also 4.8(10.1/3), this rule needs an “any part”.

What is the accessibility level of an access discriminant? 3.10.2(12.1-3/3) defines that.

Tucker's point is that this level is not necessarily the same as the one for the return object itself (although they **usually** are the same).

“of the function” should be “of the return statement” (in 6.5(21/3)).

Randy wonders if access discriminants and anonymous access are consistent. They are defined by laundry lists (3.10.2(12.1-3)) and 3.10.2(14.5)); hopefully they are consistent.

Steve would like an AARM note that build-in-place does not affect accessibility checks.

Steve wonders if the example CW_Return always was detected using the AI05-0051-1. Tucker says that this example is handled by existing wording; after an explanation, the group agrees.

Steve thinks that 6.4.1(15.1/3) needs some wording that works like the access discriminant cases – it is not just the level of the function result. Tucker suggests using “as determined by the point of call (see 3.10.2)”.

We should add italics to “determined by the point of call” in 3.10.2, and index the term in the two places it is defined. Randy wonders if this term is well-defined for calls that do not involve access discriminants or anonymous access results. It needs to be usable for explicitly aliased parameters.

We need to define “determined by the point of call” properly.

Steve wonders if we should except certain return types from this check. Randy worries that there would be a (small) maintenance hazard, in that checks might start getting applied if the type was changed. These checks only can fail if used in an allocator.

That isn't very likely, so we'll try to keep the checks.

Tucker worries that explicitly aliased checks can't be made for subprograms where the outer function doesn't have explicitly aliased parameters. Something like

```
function F (A : aliased Integer) return Integer;

function Foo (A : Integer) return Integer is
begin
   return F(A);
end Foo;
```

Randy suggests that we only propagate “point of call” if the enclosing function has an explicitly aliased parameter or an access discriminants, etc.

Note that the checks are different between

```
   return F(X);
```

and

```
   C : constant T := F(X);
begin
    return C;
```

in the first case the level of the caller is passed to F, in the second case the level is the current level.

Pass in run-time context (including the accessibility of the point-of-call) if:

- May have access disc part, include class-wide.
- Has access result.
- Requires build-in-place.
- Has explicitly aliased parameter.

Steve will try this again.

Approve intent: 6-0-2.

AI05-0235-1/03 Accessibility of explicitly aliased parameters

Tucker still prefers “representing the invocation of the entity”.

We can change (back) to that.

Erhard would rather we say, “Other than for an explicitly aliased parameter, “, rather than a more general exception.

Tucker thinks that the AI05-0051-1 wording disappeared from 3.10.2(19.2/3); we still need that original paragraph. [Editor's note: AI05-0142-4 specifically says that it is a replacement for the wording change in AI05-0051-1; if that's not true it doesn't have anything to do with *this* AI.]

Steve thinks that 19.2 needs to mention “return object or coextension thereof”. No, the level of a coextension of that of the return object, so we don't need to say that.

Tucker is worried about the dynamic check in the non-return object case. We previously said that there was a context passed in, so there isn't an extra cost.

Should we be allowing aliased parameters for task entries? This probably causes comparability problems. We don't allow access parameters for those. Take that question off line, Steve will research it.

Approve AI with changes: 5-0-3.

AI05-0236-1/01 Additional problem with preelaborated generics

Randy tries to explain the problem (it is explained in the !discussion of the AI).

Tucker thinks we could say “object or value” of non-static expressions. Also no elaboration of non-static constraints.

Gary asks about:

```
type CStringy is array (T) of Character;
CG : constant Cstringy := (others => 'A');
```

The expression would be non-static, so it would get caught.

Tucker says that is not quite right, because we want to allow aggregates – those are never static. Perhaps we should say that you can't evaluate non-static scalar expressions. But we also need to disallow non-static choices in aggregates.

Non static constants don't work, either.

Randy wonders if we should bother to fix this at all. Bob agrees, he says there is no major issue.

Bob notes that controlled types are allowed in preelaborable packages; these also generate code.

We begin to think that this is insufficiently broken to fix. Preelaborate never says anything about code being generated; C.4 rules cover that.

Also, fixing this is incompatible with existing code.

Bob volunteers to write this up as a confirmation of the existing rules.

Approve intent of AI: 7-0-1

AI05-0237-1/01 Ancestor types for synchronized formal types

The first set of brackets in the wording are “redundant” brackets, not a deletion.

Drop the “(rather than **limited**)”. Not everyone thinks that's a good idea.

Try to invert the new sentence:

“The ancestor type shall be a limited interface if the reserved word **synchronized** appears.”

Straw vote on removing “(rather than **limited**)”: 2-1-5 So don't do that.

Approve AI with changes: 7-0-1

AI05-0238-1/01 Split behavior at the limit

Approve AI: 7-0-1

AI05-0239-1/01 Hyphens in mode conformant

Approve AI: 7-0-1

AI05-0240-1/01 Rewordings from First Editorial Review

Add {} around the AARM note for 5.4(6).

Approve AI with changes: 8-0-0.

AI05-0241-1/02 Aspect-related restrictions

Erhard thinks the wording should be “no use of”. But this is similar to the first two existing restrictions.

Should No_Implementation_Aspects cover pragmas or attributes? No, this just should disallow aspect_specification for implementation-defined aspects. Users have the other restrictions to disallow the other uses.

In the second one, put an underscore in aspect_specification.

There is confusion between “No_(Aspect_Specification)” and “No_Aspect_(Specification)”.

Tucker suggests naming these restrictions No_Implementation_Aspect_Specifications and No_Specification_of_Aspect. That seems better.

Approve AI with changes: 7-0-1.

AI05-0242-1/01 No_Implementation_Units restriction

Approve AI without changes: 8-0-0.

Do we want any additional restrictions beyond this one?

Tucker suggests is “No_Implementation_Identifiers”, covering System, Standard, Interfaces, and Implementation packages in Queues.

Bob is unconvinced that this covers all of the cases; we would need to search the standard to find such places.

Randy wonders if “Long_Integer” is an implementation-defined identifier.

Tucker will go look and see if there any other cases, and he will write up a proposal.

Tucker (later) suggests requiring Long_Integer to exist in Standard. That seems OK, although it does require 32-bit integers. That doesn't seem to a problem – if an implementation really has a problem with 32-bit integers, they can rely on 1.1.3(6). But we want to encourage all implementations to have them.

There is a similar accuracy requirement for Long_Float, so we should require that as well.

Those changes would except Long_Integer and Long_Float from this restriction, but all of the other possible names (Short_Integer, Byte_Integer, etc.) are covered by this restriction.

Tucker will also propose a Profile (No_Implementation_Extensions). That would bundle all of the No_Implementation_xxx restrictions.

Those will be in a new AI.

Approve intent for this new AI: 7-0-1.

AI05-0243-1/02 Clarification of categorization

Wording for pure: “set” => “specifies”.

Drop the “; the aspect can also be set by the aspect_specification of the library_item”. We re not going to emphasize multiple ways of doing things.

Some are uncomfortable with the "declared pure"/"pure" wording.

Bob notes that we need two names here; if we drop “declared pure” (now “Pure”). Then “pure” (for the properties) would need to be renamed. “pure-eligible”, “purable”, “potentially pure”, “shall meeting the requirements of purity”.

Either that or change to “specified pure”.

Jeff suggests “stateless”. “meets the requirements of pure”.

The question: leave this as allowed, delete the answer to the question.

This AI should be Amendment.

Should “categorization pragma” and aspect get moved to 10.2.1? That would make sense.

Tucker has a wording suggestion: He notes that a library unit is not the same as a compilation unit.

Add:

A pure library unit is one where the Pure aspect is true.

A pure compilation unit is …

The “pure library unit” handles “declared pure”; “pure compilation unit” is pure.

But the big paragraph still gets clunky.

A limited view is not a library unit – 10.1.1(12.5/2). So it is OK to talk about library units.

We could say that a limited view is a pure compilation unit.

Tucker has better wording (which he e-mailed). Randy notes that we'll need an AARM note to explain. [Editor's note: There are more than 30 occurrences of "declared pure" in the AARM; it does make sense to try to change that term.]

Approve intent of AI: 7-0-1.

AI05-0244-1/01 Ancestor parts that are qualified or parenthesized

Steve sent wording after the deadline [which is in version /01 - editor].

Tucker would like “or” to be “nor” between these bullets.

Change “thie” to “this” in the second bullet.

“which” should be “that” in the last sentence.

Add a colon after the “be”.

Approve AI with changes: 6-0-2