Minutes of Electronic ARG Meeting 62M

18 November 2021

Attendees: Steve Baird, John Barnes (leaves at 12:32), Randy Brukardt, Arnaud Charlet (left around 11:35), Jeff Cousins, Gary Dismukes, Bob Duff (arrived at 10:38), Brad Moore, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Tullio Vardanega, Richard Wai.

Observers: None.

Meeting Summary

The meeting convened on Thursday, 18 November 2021 at 10:36 hours EST and adjourned at 13:30 hours EST. The meeting was held using Zoom. The meeting covered much of the agenda.

AI Summary

The following AIs were approved:

AI22-0002-1/03 Nonoverridable aspects must be primitive (12-0-0)

The following AIs were approved with editorial changes:

AI22-0001-1/01 Presentation issues in Ada 202x submission (13-0-0)
AI22-0003-1/01 Vestigial rule about inheritance of user-defined literal aspects (11-0-0)
AI22-0006-1/01 Two-pass iteration of array aggregates (12-0-0)
AI22-0007-1/01 Discriminant checks for aggregates might need to be done early (11-0-0)
AI22-0010-1/02 Predicates on private extensions (12-0-0)
AI22-0011-1/01 Reduction expression issues (11-0-0)
AI22-0012-1/01 Incompatibility with AI12-0412-1 (10-0-2)
AI22-0018-1/02 Easy fixes to Ada 2022 (13-0-0)

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

AI22-0008-1/01 Nominal subtype of a delta aggregate (11-0-0)

The following AIs were discussed and assigned to an editor:

AI22-0004-1/01 Permissions of 4.1.4 and No_Implementation_Attributes
AI22-0013-1/01 Pragma after a final label

The intention of the following AIs were voted, but then were discussed again later in the meeting (the final results are above):

AI22-0002-1/02 Nonoverridable aspects must be primitive (13-0-0)

Detailed Minutes

Welcome

Steve welcomes everyone.

Apologies

Justin sent an apology. Arnaud says that he has to leave in an hour for another meeting, and that Claire and Raphael cannot attend at all.

Previous Meeting Minutes

There were no comments on the minutes: Approve minutes: 12-0-0.

Date and Venue of the Next Meeting

Randy proposes Thursday, January 27, 2022 for our next meeting, it is 12 weeks from today. Jeff says he can't come on that day. Tucker announces that he will be out of the country. After a brief flirtation with Groundhog Day, we settle on Thursday, February 3, 2022, 10:30 AM EST.

As noted last time, we have an in-person meeting scheduled, if possible, on June 17-19, 2022, Ghent Belgium (in conjunction with Ada-Europe).

ACATS Tests for Ada 2022

We've received a few tests (from Steve, Richard, and Claire). Tucker says he has some but they need more work. Several others chime in with similar tales. We remind everyone that they've promised to do this by today.

Working Group Report – UCI (User Community Input)

Richard outlines the three stage model, with stage one being unstructured and mostly open to the public to generate ideas and possible solutions; stage two being working groups to work on specific proposals, including wording proposals; and stage three being an ARG process similar to our existing processes to refine and integrate proposals.

The group is still working on selecting platform/technology choices. We want to avoid vendor lock-in, so we are trying to consider platform/technology choices that would allow changing if necessary. We're planning to prototype on Github for stage 1, and using dedicated Google Docs folders for stage 2.

The group will have another meeting on December 3rd.

Tucker thanks Richard for his work to date leading this effort.

Unfinished Action Items

There are two unfinished action items (Steve Baird, AI12-0016-1; Tucker Taft: OpenMP Technical Report). We did not discuss these.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI22-0004-1

Randy Brukardt:

- AI22-0008-1

Editorial changes only:

- AI22-0001-1
- AI22-0003-1
- AI22-0006-1
- AI22-0007-1
- AI22-0010-1
- AI22-0011-1
- AI22-0012-1
- AI22-0018-1

Tucker Taft:

- AI12-0346-1 – also, construct the Technical Report suggested by this AI.

- AI22-0013-1

Detailed Review

The minutes cover detailed review of Ada 2022 AIs (AI22s). The AI22s 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 working Ada 202x AARM, the number refers to the text in draft 32 of the Ada 202x AARM (the submission draft). Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final consolidated Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2022 AIs

AI22-0001/01 Presentation issues in Ada 202x submission

Randy explains that these are all small things that have no semantic change and could be treated as typos in the Ada 202x draft. As such, we could fix these in the final publication version of the Ada 202x draft.

Gary notes in question (4) the second of the nonoverridables is missing an 'r'. Brad notes that there is a similar mistake in the !discussion.

The group agrees that these changes should be made in the publication version of Ada 2022, as they all represent typos (missing or extra words, punctuation, etc.). Thus, this should be moved to an AI12. [This will be AI12-0437-1 - Editor])

Approve AI with changes: 13-0-0.

AI22-0002-1/02 Nonoverridable aspects must be primitive

Tucker explains the AI. The summary isn't quite right, it could be class-wide, or it must be primitive.

You could rename a subprogram from a nested package to make it primitive. So there isn't much restriction in a package specification. Tucker notes that this might happen if you wanted to use a subprogram declared in a generic instance given after the type declaration. The renaming technique would provide a work-around, as would an expression function (for functions).

Gary wonders about "part" in the 13.1.1 wording. "part" → "a component of an aspect that is an aggregate". There is concern over using "component" here. Someone suggests using "element" instead. We do that.

Gary notes the hyphen in "non-primitive" should be dropped.

Tucker thinks the parenthetical remark is now too long. Make it a second sentence. Tucker will wordsmith that and send to Randy.

Jeff: in discussion paragraph starting "Second", "for" is unnecessary.

We also will reword the summary.

Jeff would like to see the changes, so we won't vote to approve now.

Approve intent: 13-0-0.

AI22-0002-1/03 Nonoverridable aspects must be primitive

Tucker sent these rewordings in the Zoom chat window. [Editor's gripe: cut-and-paste doesn't work from my laptop using Zoom to my desktop where I'm taking notes. No matter how many times I grab the different mice. :-)]

For new paragraph after 13.1.1(18.3/5):

> For a nonoverridable aspect of a type T that denotes one or more subprograms with a parameter or result of type T or access T, all of the denoted subprograms shall be primitive for T. The same restriction applies to elements that denote subprograms within a nonoverriddable aspect that is in the form of an aggregate.
>
> AARM Reason: Nonoverridable aspects can only be confirmed on inheritance, and a name that denotes a nonprimitive subprogram continues to denote the original subprogram, thus there would be an inherited reference to a subprogram that operates on the parent type rather than the derived type. An example of an aspect that has elements that denote subprograms is the Aggregate aspect.

Replace the !summary with:

> Nonoverridable aspects (or elements thereof) that denote subprograms with a parameter or result of the associated type T or access T must denote primitive subprograms.

Approve AI: 12-0-0.

AI22-0003-1/01 Vestigial rule about inheritance of user-defined literal aspects

We seem to be requiring overriding of a nonoverridable aspect. Delete 4.2.1(14/5).

Tucker notes that "satisified" is misspelled in the last large paragraph of the question.

Approve AI with changes: 11-0-0.

AI22-0004-1/01 Permissions of 4.1.4 and No_Implementation_Attributes

We discuss which answer we actually want. Randy notes he changed his mind; the idea is to eliminate implementation-defined attributes, and this does not do that. Tucker notes that the fixed point attributes allowed by this permission have implementation-defined semantics, which is precisely what this restriction is about.

Bob is concerned about implementation cost. Randy comments that the check has to be done after resolution, not purely syntactically, but since almost all attributes resolve without context (including all of the attributes covered by this permission), the check can be put into many places in a compiler.

Bob says that for him, the cost benefit ratio is hard to define, as both the cost and the benefit are near zero.

Straw poll, in favor of having No_Implementation_Attributes apply to attributes allowed by these permissions: 8-1-2. Oppose is Bob, Abstainers are Gary and Ed.

So we want to rewrite this AI with the opposite answer. Steve notes one could change the proposed AARM note to say that the restriction revokes those permissions, and possibly make it normative. Not sure if that is the right approach.

Steve will take the AI and do the needed rewrite.

AI22-0006-1/01 Two-pass iteration of array aggregates

Steve explains the AI. The original idea was to not bother with filters in the first pass, so it is only a concern if the second pass has too many elements.

But it turns out that allocating too much memory is a problem in some circumstances, especially with limited build-in-place element types.

So we want to require that the two passes generate exactly the same number of elements.

Tucker notes that the original idea was premature optimization, and it could have wasted of a lot of space, if the filter eliminated a large percentage of the elements.

In !recommendation, “Ada 2022” needs a space. Several people note that we will be living with this for many years, let's get it right now.

Last sentence of 4.3.5(20.4/5) change to:

> ...before accessing any nonexistent element of the array object.

Richard is worried that the wording suggests that the check isn’t made if this never happens. Tucker suggests (after a couple of false starts):

> As part of this second iteration, a check is made that it results ...

Bob agrees with this change.

Last sentence of 4.3.5(20.4/5) change:

> ...before any attempt to access any nonexistent element of the array object.

Gary has a typo at the end of the discussion “thst”.

In !ACATS, Jean-Pierre notes that we have “for to check”. Randy notes there is a lot missing here. “ACATS C-Tests are needed to check...”

Approve AI with changes: 12-0-0.

AI22-0007-1/01 Discriminant checks for aggregates might need to be done early

Steve explains the problem.

Tucker says the wording should have no “shall”. Use instead:

> {Any discriminant check is performed before the initialization of any nondiscriminant component of the aggregate object.}

Approve AI with changes: 11-0-0.

AI22-0008-1/01 Nominal subtype of a delta aggregate

The answer to the question should be “(They don’t have nominal subtypes.)”.

Or “Does a delta aggregate have a nominal subtype? (No.)”

The summary should say that the wording needs changing “nominal subtype” to “applicable index constraint”. It also has “agrgegate”.

Randy should rewrite the !summary, !question, etc.

Approve AI intent: 11-0-0.

AI22-0010-1/02 Predicates on private extensions

Steve explains the rule. Note that there are no requirements on the **subtypes** of the ancestors, and the parent type doesn’t even have to be the same type (it just has to be a descendant).

This is a rule about which predicates apply to the ancestors of the private extension. Predicates that directly apply aren’t involved. Tucker would like to add that to the rule.

> All predicate specifications that apply to the ancestors of a private extension shall also apply to the ancestors of the full view.

A second try:

> Each predicate specification that applies to some ancestor of a private extension shall also apply to some ancestor of the full view.

Steve would like his privacy example in the !appendix.

Gary has a nit on the !discussion. There isn't a C2, so should it say "could". Randy thinks that this is referring to the constant C in the question. So change C2 to C.

Approve AI with changes: 12-0-0.

AI22-0011-1/01 Reduction expression issues

In question 4, "it's" should be "its".

Tucker suggests changing Accum_Type to Accum_Subtype, then we don't need to repeat that all over. Specifically, we would no longer need "subtype Accum_Type" wording all over. Value_Type also would be Value_Subtype.

AARM Ramification 24/5: ... object {are} determined ...

Approve AI with (massive) changes: 11-0-0.

AI22-0012-1/01 Incompatibility with AI12-0412-1

There is existing code where these sorts of things are not static, but still should work. So this rule is a problem.

Steve wonders if "includes" is nontextual: does it cover default parameters? Yes, default parameters are implicitly part of a call

Gary does not like the !question. Does not like "If", Tucker suggests "Suppose".

Randy has issues with the second paragraph of !discussion, the "not" appears too late and is rather buried. Gary notes "Since remember" is also in this text. Redo it as follows:

> Note that this situation can occur both when the Pre'Class or Post'Class is first resolved, and at the time that an abstract routine is *bound* (even if the resolved routine is not abstract). Remember that this is talking about the nominal NT, but ultimately one can bind to a derived type with an abstract version of one of the included functions.

Add a note to remove the AARM Discussion about the incompatibility.

!ACATS Test typo "to ensuring".

Approve AI with changes: 10-0-2. Gary, Jeff abstain.

AI22-0013-1/01 Pragma after a final label

Randy and Tucker were discussing this. It seems to be a more general problem, and more wording would be needed to finish the proposed solution, So Tucker suggested just adding a rule at the end of the pragma section.

In the !appendix, the last message should be labeled as coming from Tucker Taft.

We discuss if we need to be more specific on the constructs involved. We focus way too much on whether components are elaborated (they are). Steve notes that pragma Assert is not allowed there, but that does not cover implementation-defined pragmas, or any other pragma that contains something dynamic.

Tucker will rewrite this AI to reflect this model.

AI22-0018-1/02 Easy fixes to Ada 2022

Question (1) has an extra “do not” in it (Jean-Pierre). He also suggests moving the parenthesized part to the end. Tucker says that’s better and the parentheses should be dropped in that case.

Tucker doesn’t like 4.3.5(39/5) wording. Just change “{each}[every] iterator_element_association{, if any,}”, and drop the parenthesized insertion.

Tucker suggests moving (4) to AI22-0001-1 and making it now. That seems like more than a typo to Randy. We discuss how far we can “stretch” typos. In the end, we don’t change anything.

Brad wonders if “otherwise” is needed in the new wording appended to 13.1(0.1/3). Steve says that you don’t want this specification to end up in a circular loop, as this rule is also a form of specification. We want this to apply only when nothing else does, and not count itself.

Approve AI with changes: 13-0-0.