Minutes of ARG Telephone Meeting

26 February 2015

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins, Robert Dewar, Gary Dismukes, Bob Duff, Brad Moore, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft.

Observers: None.

Meeting Summary

The meeting convened at 12:05 hours on Thursday, 26 February 2015 and adjourned at 14:20 hours. The meeting covered the bulk of the agenda.

AI Summary

The following AIs were approved:

AI12-0038-1/07 Shared_Passive package restrictions (7-0-3)

The following AIs were approved with editorial changes:

AI12-0135-1/03 Enumeration types should be eligible for convention C (10-0-0)
AI12-0151-1/01 Meaning of subtype_indication in array component iterators (10-0-0, question 1 only)
AI12-0152-1/01 Ambiguities in raise expressions (8-0-2)
AI12-0154-1/01 Aspects of library units (10-0-0)

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

AI12-0129-1/02 Make protected objects more protecting (7-0-3)
AI12-0138-1/05 Iterators of formal derived types (9-0-1)
AI12-0155-1/00 Freezing of operations of incomplete types with completions deferred to a body (10-0-0)

The following AI was discussed and placed into promising status:

AI12-0144-1/01 Make Discrete_Random more flexible (10-0-0)

The following AI was voted No Action:

AI12-0153-1/02 Default value of Boolean-valued aspects (8-0-2)

Detailed Minutes

Date of Next Phone Meeting

At the end of the meeting, we determined that we did need another, shorter call to finish up AIs for the Corrigendum. (There are two AIs that need rewriting, one that needs an initial draft, and one that we didn't get to.)

We decide on March 26th, noon eastern time.

Corrigendum Review

Randy notes that we should do some sort of review of the Corrigendum document. That document itself isn't that important (it just a list of changes), but it would be valuable for people to look at the result of the AIs in context (only the editor has done that with any regularity).

He suggests a review similar to the one we did for Ada 2012, where each ARG member is assigned a roughly equal group of changes. Tucker suggests that each set of text get two reviewers.

Current Action Items

The combined unfinished old action items and new action items from the meeting are shown below. (* indicates an AI that we'd like to put into the Corrigendum.)

Steve Baird:

- AI12-0016-1
- AI12-0020-1

- AI12-0129-1*

Randy Brukardt:

- AI12-0112-1 (with Ed Schonberg)
- AI12-0155-1
- Split questions (2) and (3) from AI12-0151-1 (see that discussion).

Editorial changes only:

- AI12-0135-1*
- AI12-0144-1
- AI12-0151-1*
- AI12-0152-1*
- AI12-0154-1*

Brad Moore:

- AI12-0119-1

Erhard Ploedereder:

- Compile all of the examples in the Standard (see discussion of AI12-0134-1) and report on any difficulties.
- If possible, acquire some examples where Ada containers are too slow compared to other containers (for AI12-0111-1).

Ed Schonberg:

- AI12-0002-1
- AI12-0058-1 (with Van Snyder and Tucker Taft)
- AI12-0112-1 (with Randy Brukardt)

Van Snyder:

- AI12-0058-1 (with Ed Schonberg and Tucker Taft)

Tucker Taft:

- AI12-0058-1 (produce wording, with Ed Schonberg and Van Snyder)
- AI12-0064-1
- AI12-0079-1
- AI12-0111-1
- AI12-0138-1*

Detailed Review

The minutes cover detailed review of Ada 2012 AIs (AI12s). The AI12s 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 5 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0038-1/07 Shared_Passive package restrictions

This is privacy-breaking, alternative solutions are severely incompatible. Compatibility concerns trump the principle.

Approve AI: 7-0-3.

AI12-0129-1/02 Make protected objects more protecting

Remove the redundant marking as noted by the note referencing AI12-0153-1.

Tucker would prefer that the barriers get evaluated, because this is now allowing side-effects in these functions to have defined semantics. It would be weird to allow side-effects to work, but not update the queues afterwards.

Steve will take this back to change the wording for entry barriers.

Brad asks whether **in out** parameters are now allowed for these protected functions. That's already allowed of all protected functions. The protected object is still an (implicit) **in** parameter, but the other parameters can be anything.

It might be useful to have some way to have functions in which the protected object is not a constant (and vice versa, procedures where the protected object is a constant), but that would require some additional language design. It's not something to be done in the Corrigendum.

Jean-Pierre worries if this is too much of a change, especially for the Corrigendum. Randy notes that most implementations would have had to do nothing at all for the previous proposal. But it should not be difficult to reevaluate barriers after a protected function call.

Straw vote on adding evaluation of barriers: 6-1-3

Approve intent of AI: 7-0-3

AI12-0135-1/03 Enumeration types should be eligible for convention C

“same literals” -- “corresponding literals”. Since we can't ensure that the names are the same.

Promote to binding interpretation.

Approve AI with changes: 10-0-0.

AI12-0138-1/05 Iterators of formal derived types

Tucker explains what he did. He essentially moved all of the wording into 13.1.1 so it can be shared.

He notes that the exact term to use has been controversial. So he reads all of the rules using “Mr X aspect” as the term in question. (This provided quite a bit of merriment for the editor.)

For the second sentence of the third paragraph of the 13.1.1 wording:

If a {full} type with a partial view inherits a Mr X aspect...

Otherwise that seems to be a tautology (it is not, but one has to read it three times to figure that out).

Other than that, the rules seem fine.

Turning to the term. Randy and Tucker suggest that “irrevocable” is better than “unalterable”. Jean-Pierre suggests “final”. Some groans ensue.

We take a straw vote for their favorite term:

> “irrevocable”: 1 (this seemed to be be many people's second choice)
> “unalertable”: 1
> “inalertable”: 2
> “nonoverridable”: 4
> “irreplaceable”: 1
> No preference: 1

Straw vote between the top two vote getters: “unalterable” 4, “nonoverridable” 6

We declare “nonoverridable” the winner. [My spellchecker offers “nonrecoverable” and “nonbiodegradable” for this term. Grin. - Editor.]

Should we adjust the rest of standard to make sure that this is not a problem? Steve says that the term is well-defined, it can be used as a stand-alone term. Tucker would rather take the AI back, because we need to make sure that the term isn't too overloaded. Ed in particular is concerned about conflicts between the aspect and the subprogram it names: the aspect is nonoverridable, but one still can override the subprogram named by the aspect. So vote intent and Tucker will get the AI again.

Approve intent: 9-0-1

AI12-0144-1/01 Make Discrete_Random more flexible

Steve wonders what happens if the range is null. It needs to raise Program_Error in that case.

In summary, “an” should be “a”.

“Solitaire” is spelled wrong in the discussion.

Several people note that this has come up in their experience.

Should be in Corrigendum: 4-2-4

Probably loses.

Vote promising: 10-0-0.

AI12-0151-1/01 Meaning of subtype_indication in array component iterators

Most people think the anonymous access case is silly. The argument is that since it is anonymous, naming it doesn't help any, you can't use it later.

Straw vote on (2): In favor: 2; Against: 6; Abstain: 2.

Steve says that he doesn't think it belongs in the Corrigendum. He would reconsider if we are talking about a revision.

(3) seems like a new feature. It is suggested that we make this (2 and 3) into an Amendment AI.

Approve AI (just question 1) with changes: 10-0-0.

AI12-0152-1/01 Ambiguities in raise expressions

Randy tries to explain the changes.

Tucker suggests that the subject line somehow include the Ada 2005 ambiguity. (That is subtype indications with digits conflicting with progenitor syntax in a derived type declaration.)

11.3(2.1/4) Tucker doesn't like “surrounded”, he reads it as “directly surrounded”.

Try:

> If a raise_expression appears within the expression of one of the following contexts, that raise_expression shall be within at least one set of parentheses:

Several people wonder precisely where we're requiring parentheses. We're only requiring enough to eliminate the ambiguity. There has to be a right parenthesis somewhere between the raise_expression and the start of the surrounding context. We don't care where the left parenthesis is at all. In sensible uses, the raise_expression will be last in a chain of logical operations, so the right parenthesis has to go directly after it.

People would like to see some examples.

Randy asks whether default_expression should be changed for all cases. Tucker says that parameters are likely to get aspect_specification (the SPARK team is leaning that way), and we'd need the parentheses then, so we leave it alone.

“Membershup” in AARM Reason. Use “within” in the Reason.

Steve objects that the parentheses could be in the surrounding construct. That could happen for the aggregate case, none of the others. But there's definitely a problem in the aggregate case, as the outer parentheses of the aggregate would suffice for the rule, meaning we'd still have the ambiguity.

Tucker will try to fix up the wording to handle this.

After we discuss several other AIs, Tucker has new wording.

> If a raise_expression appears within the expression of one of the following contexts, the raise_expression shall appear within a parenthesized expression within the expression:

Steve objects that that does not allow a qualified expression to provide the parentheses.

> If a raise_expression appears within the expression of one of the following contexts, the raise_expression shall appear within a pair of parentheses within the expression:

The AARM note should give some examples where parentheses are required. Bob suggests an example of an extension aggregate, possibly with two withs.

Approve AI with changes: 8-0-2.

AI12-0153-1/02 Default value of Boolean-valued aspects

Steve tries to explain the “specifically boolean” wording. Tucker thinks it is confusing with specified.

Randy suggests not making any change, because it doesn't save any wording; it adds wording to somehow deal with Default_Value and Default_Component_Value.

Tucker wonders if Implementation Advice would be valuable. It would only apply to implementation-defined aspects, and those can do what they want. So there doesn't seem to be any point.

No Action: 8-0-2.

AI12-0154-1/01 Aspects of library units

...evaluated at {the} point where it occurs...

No one wants to answer the freezing question, because we don't need to.

Someone asks about implementation-defined aspects. They don't have to be library unit pragmas. If the implementer wants them to be almost like a library unit pragma, they can do that.

Approve AI with changes: 10-0-0.

AI12-0155-1/00 Freezing of operations of incomplete types with completions deferred to a body

Randy and Ed explain the problem. Randy notes that the instance problem that originally prompted the question is not relevant to the freezing issue, as no instance is needed to raise the question.

Tucker thinks that this is a bug in the language. We have an explicit “hole” in the freezing rules for so-called Taft-Amendment types. And we have explicit rules allowing operations like these. But we don't have a hole in the freezing rules allowing these operations.

GNAT compiles this program, but Randy notes that he didn't make a program that was runnable. Ed is certain that if it compiles, it will run as well.

Tucker suggests that we adopt a rule similar to the existing hole for incomplete types. Specifically, when the profile is frozen, it does not freeze incomplete parameter types.

Randy worries that there was some bizarre case where we needed these routines to be illegal, but the AARM notes about dispatching calls can't happen because there is (now) an explicit prohibition against declaring primitive operations of an incomplete type. He's not sure if there is any other problem.

Randy will take this AI.

Approve intent: 10-0-0.