Minutes of ARG Telephone Meeting

28 January 2015

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins, Robert Dewar (first hour and last hour), 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 Wednesday, 28 January 2015 and adjourned at 15:10 hours. The meeting covered the bulk of the agenda.

AI Summary

The following AIs were approved:

AI12-0146-1/01 Should say stream-oriented attribute (10-0-0)
AI12-0149-1/01 Type invariants are checked for functions access-to-T (9-0-1)

The following AIs were approved with editorial changes:

AI12-0003-1/08 Default storage pool for storage pools (8-0-1)
AI12-0141-1/01 Type_Invariant'Class for interface types (10-0-0)
AI12-0142-1/01 Bad subpool implementations (9-0-0)
AI12-0145-1/01 Pool_of_Subpool returns null when called too early (9-0-1)
AI12-0147-1/01 Expression functions can be declared in a protected_body (10-0-0)
AI12-0148-1/02 Dangling references (10-0-0)
AI12-0150-1/02 Class-wide type invariants and statically bound calls (9-0-1)

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

AI12-0129-1/01 Make protected objects more protecting (10-0-0)
AI12-0138-1/01 Iterators of formal derived types (7-0-2)

The following AI was discussed and assigned to an editor:

AI12-0135-1/02 Enumeration types should be eligible for convention C

The following AIs were discussed and placed into promising status:

AI12-0086-1/02 Aggregates and variant parts (9-0-1)
AI12-0143-1/01 Using an entry index of a family in a precondition (9-0-0)

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, and three that we didn't discuss.)

Jeff and Jean-Pierre say that February 25th is bad for them. We decide on February 26th, noon eastern time.

Reassigning AIs

The editor notes that there are (at least) two AIs that don't appear on anyone's homework list, and do not have a wording proposal. Tucker agrees to continue working on AI12-0079-1 (Global aspect) and Brad agrees to continue working on AI12-0119-1 (Parallel operations). It would help to see wording proposals for these (the devil is likely to be in the details for these ideas).

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, ? indicates an AI that we'd put into the Corrigendum if it was finished in time, but that seems unlikely because of the amount of work remaining.)

Steve Baird:

- AI12-0016-1
- AI12-0020-1
- AI12-0129-1*
- AI for blanket rule for default for Boolean aspects (see discussion of AI12-0129-1)*?

Randy Brukardt:

- AI12-0112-1 (with Ed Schonberg)

Editorial changes only:

- AI12-0003-1*
- AI12-0086-1
- AI12-0141-1*
- AI12-0142-1*
- AI12-0143-1
- AI12-0145-1*
- AI12-0147-1*
- AI12-0148-1*
- AI12-0150-1*

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?
- AI12-0135-1*
- 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

AI12-0003-1/07 Default storage pool for storage pools

Do we want the check on “Standard”?

Tucker thinks that this wording is a mess. Tucker is still in favor of a check, for direct visibility only.

Jean-Pierre says that he thinks it is not important enough to make a check. Robert says that there are 22 uses of things called “Standard” in their test suite, so it's not that uncommon.

Tucker will take a shot at this wording for a check.

Randy suggests eliminating the 13.11.3(6.1/3) bullet and instead updating the “otherwise”. Tucker agrees with this change.

Tucker will update the AI in the background and we'll come back to this.

AI12-0003-1/08 Default storage pool for storage pools

Later, we take up Tucker's revised wording.

Remove Brad's discussion of why we didn't adopt the rule that we are adopting. Add an AARM note to say that use-visiblity isn't possible for Standard (it would always be hidden by the directly-visible package Standard).

[Editor's note: We're missing an aspect version of the rule against confusing Standard declarations. I added the following as a separate paragraph after 13.11.3(5/3):

> Standard when used as the aspect_definition for the Default_Storage_Pool aspect is an identifier specific to an aspect (see 13.1.1) and does not denote any declaration. If the aspect_definition for the Default_Storage_Pool aspect is Standard, then there shall not be a declaration with defining identifier Standard that is immediately visible at the point of the aspect, other than package Standard itself.

Gary notes that the comma should not be deleted in 13.11.3(6.3/3).

Do we want this to be in the Corrigendum? There are no objections, so we don't take a separate vote, and just declare it as part of the Corrigendum.

Approve AI with changes: 8-0-1.

AI12-0086-1/02 Aggregates and variant parts

Tucker shortened the wording amazingly. Steve says that “(selected)” is not redundant, and thus should not be in parens. Tucker says that he doesn't care. Drop the parens.

Ed notes that keyword **record** is missing in the example.

Approve AI with changes: 9-0-1.

Include in Corrigendum: 2-5-3.

So it will remain an Amendment, status is Promising.

AI12-0129-1/01 Make protected objects more protecting

Steve says there is a problem with the wording. No_Return says that it defaults to False. We don't have that here, so we need to add it. (Copy the text from 6.5.1 for this one.)

Robert says that we should have a rule that says that all Boolean aspects default to False (unless otherwise specified); writing that all over the place is rather silly. That should be added to 13.1.1, and possibly the wording removed from all of the individual places it currently exists. Steve will create a separate AI to add such a blanket rule. Robert says that he checked all of the GNAT aspects and the language-defined ones, and they all default to False. Steve notes that the automatic use of True when the name is given in an aspect specification pretty much requires setting it to False by default. So a blanket rule makes sense. [Editor's note: but beware of the cases that gave trouble in the past, specifically Default_Value and Default_Component_Value where the type just happens to be Boolean. We don't want this new rule altering their behavior.]

Tucker is annoyed that this is a whole new subclause. It doesn't seem worth that. In any event, Tucker doesn't want this to be optional (it doesn't help if some compilers ignore it), so it can't go in Annex C. Why did we put it there at all? In Portland, some guy named Tucker Taft suggested that it goes there, and no one disagreed. Steve just followed the direction. Tucker says that must have been Tucker version 2.3, because he has no idea why he would have thought that was a good idea.

OK, if it doesn't go in Annex C, where does it go?

9.5.1(4) is the current rule that this aspect modified. It should be somewhere in there. Tucker says that since this is an aspect of the protected type, so it should go in 9.4. Steve notes that that would be a forward reference. So it does go into 9.5.1.

So it goes right after paragraph 2 (aspects typically are defined as Static Semantics), and the Dynamic Semantics part goes into paragraph 4. Something like "unless both actions are the result of a call on a protected function{ and aspect Exclusion_Functions is False for the type of the protected object}".

Steve will reword this putting it into 9.5.1.

Brad wonders if we should allow specifying it individually on functions (so some could be exclusive and some could not). It seems like a lot harder to do that, especially as internal calls could come into play. (You'd have to check the aspect for both functions to tell whether exclusion is required, and even then an internal call from a non-exclusive function to an exclusive function could cause trouble.) If you had a lot of functions in a protected type, it would add a lot of clutter. And it doesn't seem important. J-P says that we'd risk killing the whole idea with complexity; including where it is useful. We could always do that in the future if users demand it.

Approve intent: 10-0-0.

AI12-0135-1/02 Enumeration types should be eligible for convention C

Tucker discusses his wording.

He came up with 16-bits as Ada requires at least 16-bit integers, as does C.

Robert wonders how this gets mapped in COBOL. Tucker suggests that it get mapped into some integer type. For COBOL, that probably would be type Binary. Robert says that isn't consistently provided. Interfaces.COBOL has a type Binary, so the implementation better know how to map it to its supported implementation. If that's a problem, it's certainly not unique to this feature.

There should be some wording as how the mapping is performed in B.3. After all, the original question was about portable usage of Convention C on enumerations .

Tucker says that this is not well-defined on the C side, and that could cause problem. Robert notes that C defines enumerations as Int, but on C++ allows enumerations to be smaller than Int. So the mapping might have to be different. The size is the problem.

Tucker will go back and say more about the C and C++ situation; we need to know how this maps. (Whatever is said will be Implementation Advice anyway, so it doesn't have to be perfect.)

We start to vote on intent, but Tucker notes that the intent isn't known yet.

Keep alive: 9-0-1.

AI12-0138-1/03 Iterators of formal derived types

Tucker would look to put each of these into a single paragraph.

The other rules are needed to avoid privacy-breaking. Gary would like some additional discussion – there's just an example and one has to figure out the problem for themselves The summary also needs a sentence saying that we need the rules to not be privacy breaking.

Tucker would like to try to factor out more of the common wording. Steve and Randy say that the rules aren't quite the same (Implicit_Dereference is different). Randy thinks it might be possible to factor out the rules involving with tagged partial views, but there might be trouble with conflicts (especially if one tries to word it in terms of what is allowed). Tucker still thinks that it can be done. He can put his pen where his mouth is, and try to reword this AI.

Approve AI intent: 7-0-2.

AI12-0141-1/01 Add Raise Expression to Introduction

Jeff suggests making the change that the editor suggested to 57.15, but not the 57.18 change (he considers it unimportant). Tucker says the first is important, the second he has no opinion.

So add only the 57.15 change to this AI.

Approve AI with changes: 10-0-0.

AI12-0142-1/01 Bad subpool implementations

Ed suggests "(see 13.11 concerned with Erroneous Execution)". Steve thinks that it is the definition, so it should not be parenthesized.

We settle on:

> If Allocate_From_Subpool does not meet one or more of the requirements on the Allocate procedure as given in the Erroneous Execution rules of 13.11, then the program execution is erroneous.

Approve AI with changes: 9-0-0.

AI12-0143-1/01 Using an entry index of a family in a precondition

Steve says that we need to define the nominal subtype.

Replace the wording about the type with:

> The nominal subtype of this attribute is the entry index subtype.

We don't need to mention the type, since it "obviously" is the same as the type of the entry index subtype.

The requeue target shall not have an applicable specific or class-wide postcondition {that}[which] includes an Old {or Index} attribute_reference.

Other wordsmithing is considered, but Randy notes that we're looking at this out of context. So let's not do that.

Should this be in the Corrigendum? Ed says that it isn't that important. Bob would say to wait for a customer request before bothering to implement. Ed worries about the implementation cost; tasking stuff can be tricky. Jean-Pierre says that there is a name for it already in the body, so the implementation can't be that hard. Several people note that preconditions are not evaluated inside the body in general.

Approve AI with changes: 9-0-0.

Include in the Corrigendum: 1-3-5.

So it will remain an Amendment, status is Promising. [Editor's note: Actually, it was written up as an omission with a binding interpretation, so it really will *become* an Amendment.]

AI12-0145-1/01 Pool_of_Subpool returns null when called too early

Gary: about about in the !question.

"plaus{i}[a]ble" in the discussion.

Approve AI with changes: 9-0-1.

AI12-0146-1/01 Should say stream-oriented attribute

Approve AI as is: 10-0-0.

AI12-0147-1/01 Expression functions can be declared in a protected_body

There is a missing } after expression_function_declaration.

We agree that null procedures and expression functions should be handled the same.

Randy asks about null procedures and expression functions in the spec. There might be effects on protected actions. It just doesn't seem worth any complications.

We discuss whether this should be in the Corrigendum. Randy argues that it was just an oversight that we didn't allow it. There is no possible semantic problem (these are equivalent to the full bodies according to 6.7 and 6.8, and of course the full bodies are allowed).

Bob says that we should allow this in bodies immediately as GNAT already supports this. It would be rather silly to suggest that their implementation is somehow wrong today when we plan to make it OK at some point in the future.

Approve AI with changes: 10-0-0.

Include in Corrigendum: 4-0-6.

So this is in the Corrigendum (meaning it's a Binding Interpretation, which matches the current write-up).

AI12-0148-1/01 Dangling references

Steve says that we don't need to talk about parts in the equality rule. The consequences could be in an AARM note.

Tucker suggests starting with:"it is a bounded error to evaluate an expression whose result is a dangling reference. Either Constraint_Error or Program_Error is raised, or in the case of predefined equality ..., execution proceeds normally.

Randy says that he doesn't want to try to do this himself, Tucker seems to know what he wants pretty clearly. So, Tucker will try to rewrite this part.

"that this {is} better" in the editor's note.

"member[e]ship" in the AARM Note.

"finalizat{i}on".

Approve intent: 9-0-0.

AI12-0148-1/02 Dangling references

Later, we look at Tucker's new wording. Someone asks whether "might" is the normal way to describe a Bounded Error.

Randy looks up various Bounded Errors, and notes that most consequences start "If the error is detected..." In particular, the meaning of invalid values starts:

> If the error is detected, either Constraint_Error or Program_Error is raised. Otherwise, execution ...

We'll use wording patterned on this.

Approve AI with changes: 10-0-0.

AI12-0149-1/01 Type invariants are checked for functions access-to-T

Approve AI as is: 9-0-1.

AI12-0150-1/02 Class-wide type invariants and statically bound calls

Tucker explains that this is similar wording to the precondition case. The main difference is that preconditions are on subprograms, while type invariants are on types. But the formal derived type model is the same. Ed notes that GNAT is literally using that model internally; it's declaring a class-wide precondition as a generic with a formal derived type which is then instantated for the actual type (each descendant type).

Typo in the example: `function Proc (P : out Root)'` should be `procedure Proc (P : out Root);`

Tucker suggests that we need to editorial review this AI especially well.

Gary asks to drop the hyphen on "non-abstract" (several places).

Approve AI with changes: 9-0-1.