Minutes of ARG Telephone Meeting

26 March 2015

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins, Robert Dewar (until 14:00 hours), Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft.

Observers: None.

Meeting Summary

The meeting convened at 12:03 hours on Thursday, 26 March 2015 and adjourned at 14:32 hours. The meeting covered the entire agenda.

AI Summary

The following AIs were approved with editorial changes:

AI12-0129-1/06 Make protected objects more protecting (9-0-2)
AI12-0138-1/06 Iterators of formal derived types (10-0-1)
AI12-0157-1/01 Missing rules for expression functions (9-0-2)
AI12-0158-1/01 Definition of quantified expressions (8-0-2)
AI12-0159-1/02 Corrections from the Corrigendum Editorial Review (11-0-0)

The following AIs were discussed and assigned to an editor:

AI12-0140-1/02 Access to unconstrained partial view when full view is constrained
AI12-0155-1/02 Freezing of operations of incomplete types with completions deferred to a body

Detailed Minutes

Corrigendum

Randy notes that this concludes our work on AIs for the Corrigendum. He will circulate a final Corrigendum document for a Letter Ballot in a week or so, and then we can submit it to Joyce for distribution to WG 9.

Current Action Items

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

Steve Baird:

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

Randy Brukardt:

- AI12-0112-1 (with Ed Schonberg)
- AI12-0155-1

Editorial changes only:

- AI12-0129-1
- AI12-0138-1
- AI12-0157-1
- AI12-0158-1
- AI12-0159-1

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
- Help Randy with AI12-0155-1 as needed.

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

AI12-0129-1/06 Make protected objects more protecting

Bob had suggested changing wording of the paragraph that defines *exclusive*, and Randy ending producing 4 more versions. He recommends the 5th version, although he doesn't feel strongly.

Tucker notes that we should always pair "exclusive" with "protected operation".

> A protected procedure or entry is an *exclusive* protected operation. A protected function F is {an} exclusive {protected operation} if the Exclusive_Functions aspect of the protected type that contains F is True.

Several people say they prefer the third version. Make Tucker's correction to that:

> A protected procedure or entry is an *exclusive* protected operation. A protected function of a protected type P is {an} exclusive {protected operation} if the Exclusive_Functions aspect of P is True.

Steve mentions that if we made this aspect part of the subprogram, we would then have to change the definition of the internal call (if a function with exclusion, called one without exclusion, then a different lock would be needed). This should be mentioned in the discussion. Tucker suggests just saying something about needing to change the lock on an internal call.

Gary has typos in the discussion: "So {the} current version of Ada...". Second to last paragraph, "it {may}[make] take". Tucker points out the first sentence of the discussion: "According [ARM]{to RM}".

Add an example to the AI of the use. (Take the first example in the discussion and add the aspect.)

Approve AI as a Binding Interpretation: 9-0-2.

AI12-0138-1/06 Iterators of formal derived types

Tucker used capital letters for the syntactic context font.

The 13.1.1 rules should definitely be under the Legality Rules handling.

Why do we have a term “match” here? It's not textual equivalence, we're allowing different expanded names.

Ed would prefer that nonoverridding means you can't specify it at all. Tucker notes that we always allow confirming aspects. Ed grumbles but agrees.

There is concern about the implementation overhead of comparing sets of operations. We don't do anything like that elsewhere in the standard.

Tucker notes that this definition of “match” means that you're comparing the set of things in this single place that would have been denoted by the inherited name to the set of things that are denoted by the specified name.

Erhard worries about renames in this case. One can construct cases where there is a consistent set via the renames compared to the original name, but they're different sets.

What does conformance use? It uses “declarations” rather than “entities”. Steve notes that we want it to require the simple names to be the same; no renames need apply. So we change the wording to use declarations.

Gary has typos: in summary, “the value{s}”;...”{that}[which]”

In discussion: in pkg4, the type declarations for Priv are missing “new”, both for private and full.

Erhard asks why the 13.1.1 text is redundant. Because each aspect specifies it individually. He notes that it is a Standard maintenance hazard: if this property gets added to a new aspect, we'd have to remember to change this paragraph.

Jeff notes that it would still be correct, just incomplete. Is it useful? Gary and Jeff say it is, so we keep it.

Approve AI with changes: 10-0-1.

AI12-0140-1/02 Access to unconstrained partial view when full view is constrained

We decide to defer this AI to Madrid, it's not that important and it's getting near the end of our meeting window.

AI12-0155-1/02 Freezing of operations of incomplete types with completions deferred to a body

Typo in the question: “exception” => “excepting”.

Randy thought that the extra text in his added wording was not necessary. We decide to remove it.

> Freezing of a profile in these cases does not freeze an incomplete subtype of the profile.

Should we do the generic instantiation rule as well? We would like to see the wording.

Tucker suggests sending it back and look at it in the future, as it is not that important.

Keep alive: 9-0-1.

Back to Randy. Tucker volunteers to help.

AI12-0157-1/01 Missing rules for expression functions

We start by discussing the five versus six items in the lists of 4.3.3(11/2). We decide to go with separate items. Tucker notes that this wording has gotten out of hand; Randy says we probably should have written this as separate bullets, but it's unclear that really helps and it's a lot more change than needed now. So use:

4.3.3(11/2):

> ...or an explicit_actual_parameter, an explicit_generic_actual_parameter, the expression of a return statement{, the expression or aggregate of an expression function}, the initialization expression in an object_declaration, or a default_expression (for a parameter or a component), when the nominal subtype of the corresponding formal parameter, generic formal parameter, function return object, {expression function return object,} object, or component is a constrained array subtype, the applicable index constraint is the constraint of the subtype;

Steve objects to “expression or aggregate” all over the place.

Randy notes that Tucker suggested defining “expression of an expression function” as “expression or aggregate”, in the same way as is done in 4.4.

Gary suggests calling it “return expression”.

So:

> The *return expression* of an expression function is the expression or aggregate of the expression_function_declaration.

Replace “expression or aggregate” with return expression in all of the AI wording.

Add the definition after 6.8(6/3).

Consider changing the use of return expression 6.5(8.d.1/4). Probably OK (it's talking about simple return statements).

In question (1) change “parenthesis” to “parentheses” (two places).

Approve AI with changes: 9-0-2.

AI12-0158-1/01 Definition of quantified expressions

There is concern that the wording in the “unchanged” paragraph also implies full evaluation rather than short-circuit evaluation.

Change the second and third sentences of 4.5.8(6/3):

> The evaluation of a quantified_expression then evaluates the predicate for the values of the loop parameter in the order specified...

Intro text: “he” => “the”. “convinient”.

Just make the intro:

> Quantified expressions provide a way to write universal and existential quantifiers over containers or arrays.

And then we can drop the note (instead of fixing it)

Erhard suggests:

> Quantified expressions provide a way to write universally and existentially quantified predicates over containers or arrays.

Tucker says that he doesn't think “quantified predicates” means anything.

Erhard comments that “**for** E **of** Arr **loop**” is a convenient way to write a quantifier. No one much agrees with that; there needs to be a Boolean result.

After more discussion that doesn't lead anywhere, we come to understand that Erhard wants to get the word “predicate” into this description. That seems valuable, so we eventually agree to use Erhard's version with “and”.

> Quantified expressions provide a way to write universally and existentially quantified predicates over containers and arrays.

Approve AI with changes: 8-0-2.

AI12-0159-1/02 Corrections from the Corrigendum Editorial Review

We decide to look at each item individually.

(1) is OK.

(2) part 1, leave the comma in. So there is no change and that item should be removed.

Part 2, Gary says the comma shouldn't be before the “are”, doesn't feel as strongly about the added commas.

Ed suggests moving the parts of the wording:

> ...type NT are bound to the corresponding operations of type T1, in the evaluation of the invariant expression for the check on T1.

Same change to 6.1.1(38/4). [Editor's note: we just need to remove the extra comma, as the wording is already in the order suggested above.]

(3) is OK.

(4) We discuss this for a while, as the “also” is thought not to help, and the “(see below)” is weird, and Gary continues to be bothered by the comma. We eventually decide to break it up further:

> A multiway tree container object manages a tree of nodes, consisting of a root node and a set of internal nodes; each internal node contains an element and pointers to the parent, ...

(5) is OK.

(6) is OK.

(7) The exception_name, if any{,}

(8) is OK. Randy notes that this is incompatible on the margins. In particular, this bans a raise_expression from being the operand of a type conversion. No one is bothered by this.

(9) Tucker argues that this wording is necessary for the user to understand what this is about. This is a highly unusual rule.

Erhard argues that it isn't normative wording in any way. He would be OK with a user note. Several people object that a user note would be several pages away (because they go at the end); it would be too late to help.

Tucker suggests as a compromise just moving the last sentence to an AARM reason, leaving the first sentence. Erhard agrees.

Make this change in both 6.1.1 and 7.3.2.

(10) Tucker thinks this is strange. Randy notes that all of these bullets (there are a lot of them) are written this way; these need to be consistent (and this is the reason for reviewing the changes in context. The change is OK.

(11) this is just a presentation issue. We discuss why the alternatives are worse. We agree to move it.

(12) This is 6.4.1, not 6.3.1.

Randy notes that it isn't important that these are exclusive. If Program_Error is raised, it's impossible to read the value to see if it was initialized, and initializing it does not raise any exception, so it doesn't matter if both are performed. What does matter is that the initial predicate (“For a scalar type that has the Default_Value aspect specified”) is true, so there has to be some connection.

Therefore, drop the “however”.

Tucker suggests reversing the wording:

> For a scalar type that has the Default_Value aspect specified, if the actual parameter is a view conversion and either
> ...
> Otherwise, the formal parameter is initialized from the value of the actual, without checking that the value satisfies any constraint or any predicate.

Several people object that that puts the normal case at the end.

Gary wonders why we're dropping “however”. He doesn't like just “If”.

Tucker suggests starting with “Furthermore, if” instead of just “If”.

That seems to be the best that we can do without the reversal.

(13) is OK.

(14) is OK.

(15) is OK. It needs a discussion item (that is blank).

Approve AI with changes: 11-0-0.