Minutes of Electronic ARG Meeting 60A

10 December 2018

Attendees: Steve Baird, Randy Brukardt, Gary Dismukes, Bob Duff, Jeff Cousins (until 13:20), Brad Moore, Erhard Ploedereder (starting at 11:45), Jean-Pierre Rosen (starting at 12:00), Ed Schonberg, Justin Squirek, Tucker Taft, Tullio Vardanega (starting at 11:45).

Observers: Richard Wai.

Meeting Summary

The meeting convened on Monday, 10 December 2018 at 11:03 hours EST and adjourned at 14:07 hours EST. The meeting was held using Zoom. The meeting covered a number of Amendment AIs (there were no regular AIs on the agenda).

AI Summary

The following AI was approved:

AI12-0190-1/04 Anonymous functions (9-0-2)

The following AIs were approved with editorial changes:

AI12-0021-1/04 Additional internationalization of Ada (10-0-2)
AI12-0079-1/09 Global-in and global-out annotations (11-0-1)
AI12-0111-1/09 Stable containers to reduce tampering checks (11-0-1)
AI12-0112-1/04 Contracts for container operations (9-0-2)
AI12-0294-1/03 More clean-ups for Ada 2020 (8-0-3)

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

AI12-0298-1/01 The default conflict checking policy is incompatible (10-0-2)

The following AI was discussed and voted to be given Hold status:

AI12-0205-1/02 Defaults for generic formal types (11-0-0)

Detailed Minutes

Welcome

Steve welcomes everyone to the meeting.

Apologies

Jean-Pierre Rosen and Tullio Vardanega both said that they would be late.

Previous Meeting Minutes

Jeff thinks that someone changed their vote from *abstain* to *for* on AI12-0296-1. But none of the abstainers seems to care (or remember, or both), so we make no change to the minutes. Approve minutes: 9-0-0.

Date and Venue of the Next Meeting

Randy proposes the next electronic meeting to be on January 14th at 11 AM EST. No one has any objections, so that date is chosen.

Unfinished Action Items

We won't look at this in detail, but Randy asked Brad why he hadn't completed anything, as it was out of character. Brad replied that he had been waiting for the minutes, and then was swamped by other priorities in the short period between the posting of the minutes and the deadline [it was only 44 hours – Editor.]

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0191-1
- AI12-0210-1
- Create an AI to determine which language-defined types have a language-defined Put_Image with a specified value (see discussion of AI12-0208-1 during meeting #60).

Randy Brukardt:

- AI12-0262-1 (review when Brad has updated)

Editorial changes only:

- AI12-0021-1
- AI12-0079-1
- AI12-0111-1
- AI12-0112-1
- AI12-0294-1

Brad Moore:

- AI12-0242-1
- AI12-0262-1 (primary author)
- AI12-0266-1

Justin Squirek:

- AI12-0213-1

Tucker Taft:

- AI12-0262-1 (review when Brad has updated)
- AI12-0298-1
- Review non-container language-defined packages for Global contracts, and create an AI (see discussion of AI12-0079-1).

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 15 of the Ada 202x AARM. 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 2012 AIs

AI12-0021-1/04 Additional internationalization of Ada

Basically, this AI defines a bunch of nested packages in the various IO packages, along with some stand-alone packages.

Tucker asks why we have some stand-alone packages and some nested. The nested packages are just part of the whole package – they don't make sense alone. And the others would have lengthy names if children: Ada.Directories.Wide_Directories. Moreover, that would be inconsistent with the handling of other Wide and Wide_Wide packages.

Jeff notes that Direct_IO.Wide_Wide has InOut_File, should be Inout_File.

We discuss the note in 11.4.1(19). Randy reminds the group why we can't change Exception Message: the type of the message appears in syntax ("raise ... with <msg>"). Additionally, some users have streamed data into the message in order to pass data along with an exception, automatically encoding the messages would break that. So the best we can do is to tell users to do the encoding themselves if they want it (possibly using Ada.Strings.Encoding.To_UTF8), and then decoding it on use. That has to be a project-specific choice. Thus we have a user note to explain.

Tucker dislikes the use of "recommend" in this wording. He suggests changing the note to: "UTF-8 encoding (see A.4.11) can be used to represent non-ASCII characters in exception messages."

Approve AI with changes: 10-0-2 (Steve and Bob abstain).

AI12-0079-1/09 Global-in and global-out annotations

The major change to this draft is that Tucker removed the Reachable stuff.

Randy asks if inherently mutable constants are covered by these rules. (Those that are immutably limited or have a controlled part – 3.3(13/3)). Tucker says that the rule allowing the use of nonpreelaborable constants covers that. He continues that there has to be some access value to do such a modification, so that has to appear in a Global.

Tucker would like to drop ", which is ignored for the purposes of the remaining rules". Randy notes that he thought that it was intended to mean that "variable" means "variable and constant pre-elaboration". Tucker explains that this was intended to allow but not require such constants. Randy is dubious that this will work.

Tucker would prefer to change the access rule to "object":

> ACCESS subtype_mark identifies the set of (aliased) objects that can be designated by values of an access-to-variable type with a designated subtype statically matching the given subtype_mark.

This covers all of the objects accessible via an access-to-object (even though the rule says "access-to-variable").

But task constants also need to be covered. They can change "value" asynchronously.

Tucker suggests adding that to the description at the very top:

"The Global aspect identifies the set of variables (which, for the purposes of this clause includes all task objects) ...".

Randy will take off-line the question of whether parts of global constants are covered properly in the legality rules. (Specifically, whether such parts are required in Global contracts when needed.)

Gary has a pile of editorials:

- The first paragraph of 6.1.2, remove the second "for a".
- "non[-]generic", "non[-]preelaborable", "side[-]{ }effect"
- "/global_/attribute_reference[']s", "global_name[']s"

Gary is finally done.

We discuss whether this AI is ready to approve. Tucker suggests that this mechanism is essentially impossible to check manually; we'll need implementations and practice to see if it has problems. So we shouldn't be afraid to approve it ,as we have to expect that with something large and new that some fixes will be needed.

Approve AI with changes: 11-0-1 (Gary abstains).

Someone needs to go through the RM to add these for the language-defined packages other than containers. We start talking about Text_IO. Ed says it would be global => all. Erhard says that is too global; it's not going to touch any global user objects, surely. Tucker suggests that we say that it is **in out** Text_IO (that is, its own state). Randy notes that there is going to be a permission to add implementation-defined packages so long as their global data is synchronized. Tucker suggests that this is a overall permission, Randy says that it should be at the start of Annex A (where there are other, similar permissions).

Someone asks about other contracts. We don't currently intend to add preconditions or postconditions to existing packages other than the containers. It would be a good idea, but it would seem to be too much work at this point. Randy did this exercise for Nonblocking (see AI12-0241-1), and he says that he isn't dumb enough to repeat that.

Tucker will take this action item.

AI12-0111-1/09 Stable containers to reduce tampering checks

Randy notes that last time there was discomfort with the duplication of specifications between the “base” package and the stable nested package. So he prepared three alternative wordings for this AI. Option 1 is the description as it is found in the AI. Option 2 is his first attempt at English wording, but it probably can be skipped since it has no advantages over the Option 3 wording. Option 3 describes the routines *not* included, then describes how all of the others are constructed. He included all of the types and other routines explicitly in a stub Stable nested package, and then added a comment to find the others in the wording.

The options can be found in the !appendix of the AI, the last message (sent December 6th).

Randy notes that he has changed his mind from the Lexington meeting; the option 3 wording avoids duplicating complex contracts and makes maintenance easier (changes to basic routines will not require changes to the stable package stubs or the stable package wording unless a new routine is added that tampers with elements).

Several people note that Option 3 is much shorter. Someone notes that duplication in these packages would make it harder for readers to look up a routine that they are interested in. And having the duplication always would leave the question as to what the difference is.

We agree that Option 3 is best; the AI should be rewritten in this form.

Randy asks if we want the cursor-only routines in the stable form. He notes that the contracts are much nastier as one does not know the actual container involved, meaning that they often cannot be used in parallel operations even if they would have been safe. Tucker notes that the stable nested package will be mostly used by the compiler to reduce overhead, so not having the routines is not going to be a problem. They're oddly inconsistent with the rest of the package anyway. So leave out the cursor-only routines (except of course, Has_Element).

Approve AI with (massive) changes: 11-0-1 (Ed abstains).

AI12-0112-1/04 Contracts for container operations

Randy explains that for workload purposes, he'd rather apply these changes directly to the RM. He thinks that the AI form would not be useful for reviewing the changes anyway – one cannot see the old, deleted text in the AI (while one can if using the appropriate RM format). And writing the AI for the remaining containers would be 3 times the work.

Tucker suggests trying to do these separately from other changes for best review. He also suggests saving the initial files before making any changes.

It is suggested that the AI be used to provide a sample of a complete update for one container. Randy notes that was done long ago – the AI provides the complete set of changes for all forms of Vector, as well as the Indefinite_Holder. The other containers would be similar and fairly rote.

Someone notes that the postconditions only mention Length, rather than fully specifying the result. Randy says that he only tried to specify “easy” properties; he wasn't trying to fully specify the postconditions. Is that level of postcondition OK? Several people quickly agree. We don't want to try to fully specify these here; a full specification probably would end up more complex than the existing English description. Randy notes that he only tried to get rid of the English description of trivial requirements, such as the one that says the source is empty after a Move.

Tucker wonders if we should add a permission to allow implementer to add more Postconditions. Bob says that its not needed, since that would be semantically equivalent, and implementers can always make semantically neutral changes. Randy agrees. Perhaps we should have an AARM note somewhere.

We agree to this style for the containers; Randy will do the rest of the containers the same way, but not in the AI.

Approve AI with changes: 9-0-2 (Erhard and Bob abstain).

AI12-0190-1/04 Anonymous functions

Tucker says that returning anonymous functions would be a substantial implementation burden. It seems like too much at this point.

Gary says that he doesn't see a compelling need to add this feature. Tucker says that every language that he knows of has added this recently. He adds that not all have added them to allow return them.

Ed says that this is just an invitation to say that Ada is just another incomplete language. Tucker disagrees.

Should the AI say something about not being able to return these?? We never really answer this question, as we digressed into a discussion of what happens if you try (you get an accessibility failure from trying to return a local subprogram to a more global access type).

Approve AI: 9-0-2 (Gary and Randy abstain).

AI12-0205-1/02 Defaults for generic formal types

Jean-Pierre worries that there isn't any other place where one can omit the type this way. Someone notes that we have done something like this for renames.

What would this be used for? An array index defaults to Positive, for instance. One could define a default storage pool type, and still support the use of specialized pools.

Ed Schonberg worries that there could be a significant amount of implementation work. Randy thinks that the cost would be limited to just the parameter matching for a generic instantiation. Steve comes up with the case where the default would depend on another generic formal (or instance). There is concern about the implementation cost; specifically as the benefit isn't very high.

Richard suggests that this would improve the user view of generics. It would be useful for defining secondary types that sometimes one would want different but usually would not. That could improve the flexibility of generic units without hitting the user over the head with minor options.

It is asked whether any users outside of this group have asked for this. The original 2002 proposal came from an outside user, and this proposal came from a non-ARG member at AdaCore. But it clearly only matters to people building reusable generics, which is probably a small number.

Straw vote should we continue to discuss the proposal: Stop: Ed, Steve, Bob, Tullio; Abstain: J-P, Tucker, Erhard, Gary; Continue: Randy, Justin, Brad. Some people haven't voted, but there doesn't seem to be a need to figure out the missing votes.

So No Action. Randy suggests that we put it an hold (with the other two defaults), so we can reconsider it in the future when we may have more compelling use cases.

Hold status: 11-0-0.

AI12-0294-1/03 More clean-ups for Ada 2020

Randy explains that this fixes 5 separate isssues, all of which were discussed in e-mail, and wording was agreed to there. Issue 1 came from the Lexington meeting, issue 2 came from a confusion that Brad had about this paragraph, issue 3 came from Randy's review of the wording while inserting it, issue 4 came from the Lexington meeting which thought incorrectly that it was properly handled, and issue 5 also came from Randy's rereading of the wording.

Jeff notes a typo: “Nomblocking” in the last paragraph of discussion for (4).

For issue (1), Steve wonders if requeue needs some similar legality rule. We want to allow requeue here (Tucker explains a possible implementation), and it appears that the existing rule has the right meaning in that case.

Again for issue (1) Tucker worries that this wording disallows an accept within a task body that appears inside a procedural iterator. That is weird but possible. Tucker tries to come up with wording to fix this: add “...whose entry_declaration is outside the loop_statement”.

Steve suggests that using “occur” would be more RM-like: “...whose entry_declaration occurs outside the loop_statement”.

We then argue about “is” vs. “occurs”. Several people (including Steve) decide that “is” is better because it is shorter. The editor pulls rank and says that “occurs” is more RM-like – see examples in 8.2 and 8.3.

For issue (3), Randy notes that the wording didn't make a lot of sense just reversing the order of elaboration. Tucker eventually reordered and rewrote all of these paragraphs, which also eliminated some other oddities (like talking about the creation of “the” loop parameter object).

Issue (4) is just adding a classic assume-the-worst rule and the standard generic boilerplate. The wording was moved into a Legality Rules section so the boilerplate would make sense.

For issue (5) we discuss whether the rule is clear enough for the case where there are two policy pragmas in the same declarative_part. Steve says that “innermost region” doesn't seem well-defined in that case. Tucker notes that this wording was designed to work much like Suppress. Suppress defines the “region” to which the permission applies is from the pragma to the end of the declarative region; those words are lacking here.

We decide, however, that this is hair-splitting; no one will truly be confused by this wording. However, we should add a To Be Honest after paragraph 8/5 note just to be sure, leaving the wording to the editor. It needs to say something about the fact that the “region” talked about here is the region to which the pragma applies, and not the declarative region in which the pragma appears.

Approve AI with changes: 8-0-3 (Bob, Erhard, Tullio abstain; the latter two members just arrived).

AI12-0298-1/01 The default conflict checking policy is incompatible

The !summary is unclear. “... less clear {what is}[of] the value ...”

Randy notes that he wrote up one possible solution, but there are at least 4 other possibilities noted.

Tucker says that he would prefer to keep Known_Conflict_Policy completely separate from Parallel_Conflict_Policy because they are so different. It's not really possible to be certain if KCP is a subset of PCP for parallel constructs.

Ed and Steve say that it would be easier to understand if each of these policies are a superset of the preceding policy. No one argues against that.

Tucker suggests adding Known_Parallel_Conflicts_Check and basing Parallel_Conflict_Checks on that.

Randy wonders if that would leave some users being unable to get the combination they want. (Specifically, the combination of Parallel_Conflict_Checks and Known_Task_Conflicts_Check). [Editor's note: what about cross conflicts between a task and a parallel construct??]

We decide that the task conflicts aren't as important, and we could leave those to implementors. After all, implementation-defined policies are allowed.

Tucker would like to think about this some more. So he will take this AI. He is directed to define a hierarchy of checks and to ensure that existing code has to stay legal – no checks on tasks by default. [We didn't say but surely implied that the Parallel_Conflict_Checks for parallel constructs has to remain as the default as well – Editor.]

Approve intent of AI: 10-0-2 (Jean-Pierre, Gary abstain).