Minutes of ARG Meeting 63A

5 October 2023

Attendees:

Steve Baird, Randy Brukardt, Jeff Cousins, Gary Dismukes, Niklas Holsti, Brad Moore (gone 13:00-13:20), Alejandro Mosteo (leaves 13:00), Jean-Pierre Rosen, Justin Squirek, Tucker Taft, Tullio Vardanega, Richard Wai (leaves 13:00).

Observers:

Mohammad Ali Asgar (Canada), Robin Leroy (Unicode Consortium Liaison to SC 22).

Meeting Summary

The meeting convened on Thursday, 5 October 2023 at 10:34 hours EDT and adjourned at 13:30 hours EDT. The meeting was held using Zoom. The meeting covered about half of the agenda.

AI Summary

The following AIs were approved:

AI22-0072-1/04 Meaning of direct references to components (12-0-0)

The following AIs were approved with editorial changes:

AI22-0026-1/02 Problem with nested type extension check (11-0-1)
AI22-0080-1/01 Clarify definition of declarative region (12-0-0)
AI22-0081-1/01 Allow named notation for reduction expression arguments (12-0-0)
AI22-0084-1/01 Resolution of aspects that are subprograms (10-0-0)

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

AI22-0033-1/03 Additional terms and definitions (9-0-0)
AI22-0082-1/01 Problems with nonlimited reference types (10-0-2)
AI22-0083-1/01 Treat dynamically-tagged expression as class-wide in various contexts (8-0-2)

The following AIs were discussed and voted No Action:

AI22-0034-1/02 Implementation model of dynamic accessibility checking (12-0-0)

Detailed Minutes

Welcome

Steve welcomes everyone to this meeting.

Apologies

Arnaud Charlet and Bob Duff said they would be late or miss the meeting entirely. Raphael Amiard was unable to attend.

Previous Meeting Minutes

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

Date and Venue of the Next Meeting

Our next electronic meeting is proposed for Thursday, December 14 with the usual time (10:30-13:30 EST [-5 UTC]) and method (Zoom). There are no objections to this date.

Final Ada 2022 RM

Randy says that he has finally posted the final Ada 2022 RM on Ada-Auth.org. He hasn't announced it yet as he is still working on getting it posted on AdaIC.org, and several web pages still need updating on both sites. He's also working on a first (incomplete) draft of the next Standard (known for now as Ada 202y).

ARG Plan for Future Work

Randy created and circulated a set of bullet points on this topic. No one had any changes or suggestions outside of typos.

Randy wonders if we need to formalize any part of this in the procedures.

Tucker looks at the current procedures and notes that we already have formalized prototyping in them. We agree that Bob's suggested version is too detailed for this; it's best that the prototyping group and the AI author work together to make changes to the proposal as needed. Tucker especially noted that he didn't like the implication that making implementation easy would get priority over other considerations.

Meeting rules should be informal but mentioned in the procedures. The section 1.3 about meetings doesn't mention virtual meetings (like this one) at all, so it needs updates to support those (especially as that now is the primary mode).

Tullio suggests that we write the requirements for an in-person meeting rather than the frequency: (1) Sufficient work exists to justify such a meeting; (2) All such meetings will be hybrid, so appropriate Audio/Video equipment needs to be available.

Randy will take an action item to update the procedures accordingly.

Approve intent of plan document: 12-0-0.

User Community Input Working Group Report

Richard updated the Readme. He requests review, especially by the people who previously reviewed it.

Unfinished Action Items

Ed Schonberg is transitioning to being fully retired, so he has asked that someone else take on his ACATS test. Justin had not wanted to work on his ACATS assignment as GNAT is not implementing it, so Randy suggests he take over the filters assignment from Ed. Justin agrees with the change in assignment and promised to work on it soon.

There are plenty of unfinished action items, but we did not discuss those.

Current Action Items

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

Steve Baird:

- AI22-0076-1 (with assistance from Tucker Taft)
- AI22-0082-1
- Github Issue #53 (assist Gary Dismukes)

Randy Brukardt:

- Update the procedures with additional meeting description and rules.
- AI22-0051-1

Editorial changes only:

- AI22-0026-1
- AI22-0080-1

- AI22-0081-1
- AI22-0084-1

Gary Dismukes:

- Github Issue #53 (with help from Steve Baird)

Edward Fish:

- AI22-0022-1

Alejandro Mosteo:

- Github Issue #5 (with help from Tucker Taft)
- Github Issue #61 (with help from Tucker Taft)

Justin Squirek:

- ACATS C-Test(s) for filters

Tucker Taft:

- Set up resources and recruit people to help prototype new features (both Ada 2022 and new ones) in various open source technologies.
- OpenMP Technical Report.
- AI22-0033-1
- AI22-0055-1
- AI22-0063-1 (Split work with Richard Wai)
- AI22-0075-1
- AI22-0076-1 (assist Steve Baird)
- AI22-0077-1
- AI22-0083-1
- Define "current instance" for single tasks and protected objects (see discussion of AI22-0072-1)
- Github Issue #1
- Github Issue #5 (assist Alejandro Mosteo as needed)
- Github Issue #61 (assist Alejandro Mosteo as needed)

Richard Wai:

- AI22-0063-1 (Split work with Tucker Taft)

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 2022 AARM, the number refers to the text in draft 35 of the Ada 2022 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

AI22-0026-1/02 Problem with nested type extension check

Steve was asked to create a static rule. He notes that the runtime check given in this AI is unchanged from last time; we did not spend any time looking at it this time.

Steve explains the static rule by noting that if the type of an object is too short-lived for a usage, then surely the object itself is also too short-lived for that usage. He made a rule based on that (we can check type properties at compile-time, while object properties aren't known [in general] until runtime). We add an AARM note to that effect.

We wonder why there is not a similar rule for tagged conversions. This is between a designated type and an access type; there is no counterpart with similar types. Note that we use runtime tag checks for tagged types in order to improve usability, but that doesn't apply to accessibility checks.

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

AI22-0033-1/03 Additional terms and definitions

Steve wonders whether someone should systematically go look for additional terms. We welcome additional suggestions, but we're not going to look for them.

Tucker suggests that we want to define here terms that are used in many places. Terms that are only used in a single subclause or a few closely related subclauses need not be defined here.

Tucker will take the AI to finish this version.

Approve intent: 9-0-0.

AI22-0034-1/02 Implementation model of dynamic accessibility checking

We approved the other alternative last time, so this needs to be No Action.

No Action: 12-0-0.

AI22-0072-1/04 Meaning of direct references to components

Randy notes that both of the open issues from last time don't need any change. Steve had said that entries need similar wording. But those are not objects (or types), they're entries, and the Dynamic Semantics of calls (and other operations) have specific rules for these cases. Tucker had noted that "current instance" is not defined for single protected declarations, but the wording was carefully constructed not to use that term for single protected declarations.

In addition, it turns out that much existing wording in Chapter 9 does use "current instance" in contexts that must apply to single protected declarations (and single tasks as well). So there is an existing problem with that, which we should handle in a separate AI. Tucker will take an action item to define "current instance" for single task and protected declarations.

Randy added an equivalence to Selected_Component in the new wording because many rules in the standard use that for component references. We want those rules to apply to these references as well.

Approve AI: 12-0-0.

AI22-0080-1/01 Clarify definition of declarative region

A user was confused by the wording (is it just where the declarations are [the declarative_part] or does it include the statements [the entire construct]– the latter is the case). Tucker rearranged the wording to clarify it. No semantic change is intended.

Change the classification to Presentation.

Approve AI with changes: 12-0-0.

AI22-0081-1/01 Allow named notation for reduction expression arguments

We do not allow reordering of operands to attributes (or pragmas), and that remains the case.

Tucker would prefer to only name the Initial_Value in some uses. We put square brackets around the "Reducer =>" part, and rewrite the recommendation.

Gary is concerned about using the identifiers directly in the syntax. We do that for (specific) pragmas, so there is nothing new here; we already do that in the RM.

Jean-Pierre notes that pragmas allow any combination of named and position references.

We argue about that for a while. Some people would prefer the maximum flexibility and the simplification of the grammar that would result. Other people would prefer that we ignore that pragmas are weird and do it as with calls here.

A choice is not obvious. We take a straw poll. After much attempting to get everyone to vote (not everyone did), and people changing votes in the middle, we end up with:

- Both or neither (the original AI proposal): 0;
- Reducer is optional: Randy, Gary, Richard, Steve, Tullio, Tucker, Jean-Pierre, Brad.
- Both are optional: Jeff, Justin, Niklas

So we just make the Reducer optional (as we had already done previously). We also modified the middle example to use this new option.

Approve AI with changes: 12-0-0.

AI22-0082-1/01 Problems with nonlimited reference types

Steve explains the AI. An AdaCore customer ran into an ambiguity. Tucker realized that the case would also occur in user-defined indexing, which seems like a significant problem.

This is a usability problem, not a correctness problem (the language is well-defined).

Tucker noticed that this problem goes away if the reference type is limited.

The predefined containers need to be changed (specifically, all of the reference types need to be limited). And add a recommendation to users that they do the same. That would be a note in the new Usage section.

Steve notes that we could go further and require Constant_Indexing and Variable_Indexing to be limited reference types. Further still would be to require all reference types to be limited.

Tucker notes that we looked at a preference rule, but that seemed more complex. He's unsure if there are any Beaujolais effects from this; Steve claims that there are. He starts to explain, but we cut him off as we're don't have such a rule proposed today.

Randy worries that a lot of existing code uses User-Defined indexing, and forcing people to change it (as opposed to suggesting that) is not going to make people happy. That sort of code is often third-party packages, so changing it can be difficult.

Gary wonders how making things limited helps the assignment case. It does not, but specifying both constant_indexing and variable_indexing eliminates the problem (assuming that they don't use the same reference type, which would defeat the purpose of specifying them differently, thus it is unlikely), so in the usual case there is no problem with assignment.

Tucker notes that changing just the language-defined containers is really easy for implementers. Other fixes proposed are harder (especially a preference rule, which would require mucking with a delicate part of the compiler).

Niklas notes that the example is not ambiguous unless there is a use clause for the container instantiation. Otherwise, the "=" operator for the reference type is not visible. Randy notes yet again how package use clauses are evil, which gets a lot of groans.

But Randy notes this is probably the reason that many users have not encountered this problem. If one uses expanded names, prefixed notation, and/or use type clauses rather than package use clauses, then there is no ambiguity problem. Beating a dead horse, this is why we have to limit incompatibilities to highly unusual cases.

Tucker corrects the example by adding the appropriate use clause.

The intent is to change language-defined reference types to limited. Add a Usage Note about this.

Approve intent: 10-0-2. Gary, Jean-Pierre abstain.

Steve gets the AI to update it accordingly.

AI22-0083-1/01 Treat dynamically-tagged expression as class-wide in various contexts

Tag-indeterminate functions can be called with a dynamically tagged argument (usually class-wide); they effectively act as class-wide. But they cannot be converted or renamed as if they are class-wide.

Tucker proposes to change that.

Randy had sent some wording comments and suggestions (he also included one Brad had sent). He thought Tucker would get a notification about those, but he didn't. Those need to be addressed before we approve this AI.

Approve intent: 8-0-2 Brad, Jeff abstain.

AI22-0084-1/01 Resolution of aspects that are subprograms

Randy explains the problem.

We'll leave the "unless otherwise specified", since it is harmless and we can imagine uses for such a thing (a function of a component type of an array type is suggested) in possible future language-defined aspects. Remove the editor's note about that.

We want to allow naming subprograms declared in nested packages, including in instances. Other legality rules might not allow using those, but they should be considered if appropriately visible. It's outside subprograms, such as the ones made visible by use and use type clauses, that we need to ignore.

Approve AI with changes: 10-0-0. (Richard voted Yes before leaving.)