

Minutes of the 4. ARG Meeting 11-13 April 1997, Henley-on-Thames

Attendance:

ARG Members: Erhard Ploedereder, Pascal Leroy, John Barnes, Gary Dismukes, Bob Duff, Joyce Tokar, Ted Baker, Kiyoshi Ishihata, Stephen Michell, Mike Kamrad

ARG Associates: Anthony Gargaro

Invited Guests: Laurent Guerby, Offer Pazy, Brian Dobbing

Meeting Summary:

The meeting convened on the 11th at 9 a.m.

Erhard handed out the results of the WG9 approval in December of ARG-approved AIs submitted to WG-9 and then briefly reported that AI-12 was rejected by WG9, since it lacked a summary. On the issue of uniformity, Rudolf Landwehr is unlikely to complete his WG9 action item to scan old UIs for applicability to Ada95. Consequently, the ARG would have to do this work, but ARG work is already suffering under insufficient funding. It was the consensus to not become active on creating UIs, but dealing with any that came in as Ada comments. Additionally, the vendors appear to be meeting on their own to settle uniformity issues.

Next, we discussed method and schedule for turning the AIs into a Standard Corrigendum. Erhard reported that the C Corrigendum was quite similar to a collection of AIs, i.e., a set of questions and their dispositions, sometimes but not always associated with actual wording changes to the Standard. There was an understanding that we could produce such a Corrigendum approximately by summer 1998. This would fit with an overall schedule that allowed for two corrigenda before the next revision of the standard.

The funding situation of some members was discussed and its impact on the ARG process.

The next meeting is scheduled for 14-16 November in St.Louis, adjacent with the Tri-Ada meeting. We need WG9 to move their meeting to 13 November to accommodate our meeting.

Erhard declared that all meeting attendees can participate in straw votes. For official votes only the ARG members can vote.

The following issues were discussed on the following days:

11 April: AI-58, 103, 104, 119, 126, 159, 164, 165, 166, 172

12 April: AI-67, 85, 117, 131, 133, 141, 147, 148, 162, 166, 168, 177, 179, 180, 181, 182

13 April: AI-54, 55, 60, 64, 86, 94, 98, 99, 100, 102, 105, 116, 125, 130, 135, 158, 161, 167, 183, 184

The following AIs were **approved without change**:

- AI-54 When is a Small clause allowed? (6-1-0)
- AI-60 Attributes of Fixed-Point Types (6-0-1)
- AI-64 Elaboration checks for renamings-as-body (9-0-0)
- AI-86 Passing generic formal packages with (<>) (5-0-2)
- AI-94 Exponentiation: 0.0 ** 0.0 raises Argument_Error. (7-0-0)
- AI-99 Accuracy requirements for elementary functions (6-0-1)
- AI-125 Order of Size and Small Clauses for Fixed Point Types (6-0-1)
- AI-135 Circular Renamings as Body (6-0-3)
- AI-180 Elaboration Pragmas & "Mentions" (8-0-2)

The following AIs were **approved with changes**:

- AI-55 Overflow for Adjacent, Machine, and Model attributes (6-1-0)
- AI-67 Pragma Restrictions(Max_Tasks) (7-0-3)
- AI-98 unknown_discriminant_parts on generic formal types (6-0-1)
- AI-102 A generic formal limited private type can have an access discriminant (6-0-1)
- AI-104 Version and Body_Version attributes (8-0-1)
- AI-126 Classification of Language-Defined Packages (letter ballot requested)
- AI-141 Exceptions in Interfaces.C and its Children (8-1-1)
- AI-177 Interfaces.C.Strings.Value with len returning String (4-0-6)
- AI-179 Finalization and Unchecked_Deallocation (9-0-1)
- AI-181 components of Stream_Element_Array should be ALIASED (8-1-1, lett.ballot req.)
- AI-182 Extension aggregates with controlled subcomponents (10-0-0)
- AI-183 Primitive operations declared before it is known if the type is tagged (7-0-0)

Major changes or initial writeups of summaries were agreed upon for the following AIs:

- AI-58 Accessibility Rules for Shared_Passive Packages
- AI-103 Storage pools and access types designating task types
- AI-117 Calling Conventions
- AI-119 Is Normal Termination an "External Interaction"?
- AI-133 Controlling bit ordering (6-0-4)
- AI-148 Requeue of protected entry calls (6-0-3)
- AI-159 Shared Variables in Shared_Passive?
- AI-162 Anonymous allocators and tasks/finalization
- AI-164 Definition of remote access type
- AI-165 Recursive use of task attributes isn't considered
- AI-166 Parameterless_Handler values designating default treatment (9-0-0)
- AI-167 Erroneous scalar Unchecked_Conversion?

The following AIs **failed** to reach consensus:

- AI-131 Interface to C -- passing records as parameters of mode 'in' (2-3-5)
- AI-172 Optional main subprogram? (6-5-0)

The remaining AIs were tabled without decisions to await expansions, expert reviews or initial writeups.

As part of the discussion of AI-172, the group unanimously decided to consider ACVC test LA1001F.ADA test to be an invalid test.

Finally, the group enthusiastically approved their appreciation to John Barnes for the local arrangements.

The group adjourned at 3:02 p.m. on the 13th.

Discussion Details on AIs

AI-54 When is a Small clause allowed?

It is clear that the first statement (illegal for decimal types) is intended by the language (and therefore the AI is merely a confirmation). The second statement (illegal for derived types) is supported by statements in the RM. Pascal objects to the second statement because it is not upward compatible with Ada83.

Approved, 6-1-0 (Pascal voted against the second statement.)

AI-55 Overflow for Adjacent, Machine, and Model attributes

Adding an explanation is necessary (by prompting Ken Dritz) to enhance the value of this AI because simply restating the rules in the RM is insufficient.

Approved with this proviso of an added explanation, 6-1-0 (Erhard prefers to see the results).

AI-58 Accessibility Rules for Shared_Passive Packages

Bob says that this corrects a known bug in the RM. The only change to this AI has been the addition of messages from Offer and correction of a couple of bugs that he pointed out.

The issue deals with multiple passive partitions, especially in cases where there are no explicit dependencies among shared passive library units of these passive partitions. The implicit dependencies are caused by the active partitions which use these shared passive library units. Offer argues that the original intent of the language design was primarily to prevent dangling references in shared passive library units in passive partitions that exist for the duration of program execution, and then the attempt was made to extend this model to the case where some

shared passive library units (and passive partitions) come and go, which is causing the problems; we should focus on solving the problem for the simple model of „permanent“ partitions and resort to implementation-definedness for the „come-and-go“ case. All agree that this intent could handle the problem of accessibility among shared passive library units and the normal accessibility rules for non-distributed applications can then be used and no extra rules are needed to handle the simple case of „permanent“ passive partitions.

Still there remain several questions

- Can an implementation provide the capability of passive partitions that come and go? Apparently so, but there would be additional layers of accessibility checks that the implementation would need to apply. What can the RM say about legality of accessibility and runtime checks in this case?
- Does the user need to add explicit with clauses to make sure that all passive partitions remain existent long enough? Bob says yes and cast the AI to say so.

Example diagram: [this is a diagram where there are three passive partitions and three active partitions and each of the active partitions have dependencies to two of the passive partitions, nicely cross-hatched, and access parameters in the active partitions are moving access to the passive partitions around in an arbitrary manner.] Mixing access parameters with passive partitions as shown in the diagram leads to dangling references and erroneous conditions, if partitions go away. No simple checks (such as comparison of depth indices) can catch this situation. There are several choices to solve this problem:

- Throw out objects of general access types in passive partitions - this is way overkill.
- Eliminate access parameters with values from passive partitions - this is less overkill.
- Make a special exception on accessibility rules, such that the accessibility of a package is no deeper than itself - very strange like an Escher drawing when you follow the chain of dependencies (and accessibility levels) back to the original package.

In example 3, replace X by X'access.

Much of the discussion was on narrowing the „erroneous“ cases and achieve definedness e.g. for the model of non-disappearing partitions.

Bob is asked to update the summary again to clarify the situation some more. There is reluctant consensus, though, that the intent of the AI is correct as written. A letter ballot is requested.

AI-60 Attributes of Fixed-Point Types

The request was made for an attribute that is obviously not present in the RM. The only question is whether this is really an attribute definitely expected by users in the numerics community. Does this request expose a real hole? The group could not make the case for the importance of adding this attribute. Approved as stated, 6-0-1.

Note: More input is needed from the numerics community to help assess the numerics AIs. Erhard will propose to WG9 that the next ARG meeting will set aside time at the St.Louis meeting for considering the numerics AIs. (However, as it turned out, we were able to deal with most of the numerics-related AIs at Henley, since they did not require a deep understanding of numerics issues.)

AI-64 Elaboration checks for renamings-as-body

After asking if there were any comments or corrections, Erhard asked for the vote. Approved 9-0-0.

AI-67 Pragma Restrictions(Max_ Tasks)

There is a contradiction between the two Annexes about the violations of this restriction

- _ The RT Annex only allows the program to either run or raise Storage_Error
- _ The Safety Annex requires the program to be illegal

Erhard suggests that, if the compiler can detect the violation of the restriction, it should reject the program as illegal. (He also questions the special role of the value 0 versus any other static value, where some violations could be just as easily detected at compile-time.) Bob objects that this violates a fundamental principle of the language, namely, if general violations of language rules can only be detected at runtime then an implementation is not permitted to make the program illegal, even if specific violations can be detected for every execution of the program. Bob says that we shouldn't get involved in singular AI issues to permit this behavior but, if we were to consider such a language change, we should step back and create a general AI on permitting compile-time rejection of programs for all compile-time detectable runtime violations that will surely arise. Erhard would welcome an AI on this point.

Offer observed that if No_Tasks and No_Asynchronous_Selects were added as language-defined pragmas with corresponding legality rules, then we could get the compile-time rejection we desire in this case. It would be a shame, if the „parsimony“ applied here by reusing the Restrictions pragma made the check a run-time check.

Erhard points out that this problem also applies to Max_Asynchronous_Select_Nesting. After examination, all agree it applies to both.

The group approved the AI (expanded to apply to both Max_Tasks and Max_Asynchronous_Select_Nesting), 7-0-3.

AI-85 Questions about Append_File mode

There are unfinished action items on this AI (by either Bob or Tuck) and consequently this AI is tabled for later completion by Tuck or Bob.

AI-86 Passing generic formal packages with (<>)

After close examination of the examples (which present the „hardness“ of the issue, especially on the afternoon of the last day), the summary was found to accurately describe the straightforward application of the rules. Approved, 5-0-2.

AI-94 Exponentiation: $0.0 ** 0.0$ raises `Argument_Error`

Approved as obvious, 7-0-0.

AI-98 `unknown_discriminant_parts` on generic formal types

In the last message of the AI, Bob made a recommendation for one of two fixes to 12.5(8) or 3.7(8), respectively. It was decided that the change to 3.7(8) was the simpler fix, namely replacing ‘`known_discriminant_part`’ with ‘`discriminant_part`’.

Approved, 6-0-1, with an appropriate write-up for the fix to 3.7(8), including replacing the semi-colon with a period.

AI- 99 Accuracy requirements for elementary functions

Approved as obvious, 6-0-1.

AI-100 Truncation required if `Machine_Rounds` false?

Tabled due to the lack of expertise present.

AI-102 A generic formal limited private type can have an access discriminant

This does not appear to be a ramification because it is a very straightforward result of the RM, hence confirmation is recommended. Also the Scribe typo should be removed. Approved with these changes, 6-0-1.

AI-103 Storage pools and access types designating task types

Stephen objects to this capability (from his high integrity viewpoint).

Offer said that the intention for allocating a task's stack from a storage pool for an access to a task type didn't have TCBs and stacks in mind, nor any other data structures that the RTS uses for tasks.

Still a user should be able to defined a storage pool mechanism for handling the user-defined types that include task objects.

The current wording of the summary does not support the intention. Instead, the question in the AI better reflects the original intention. Consequently, Erhard recommends that the summary wording be derived from the question section.

Tabled. The suggestion to discuss the wording over lunch and revisit the AI later was not taken up.

AI-104 Version and Body_Version attributes

Erhard asks the distributed system folks at the meeting whether they have any insight on this issue as the ARG has not reached a consensus or even a common understanding of the usefulness of these attributes.

Several interpretations of the attribute are put forth:

Offer: 'Version can be used by client partitions to check the consistency of services they call in server partitions (where they are in „normal“ packages); also it can be used by a partition to create internal version information that it can package for service requests.

Anthony: 'Version is used by separate tools, like ADEPT, to check the version consistency of the units in an distributed application.

Offer and Ted: 'Version is not necessary for an implementation that checks the consistency at (static) link time.

Bob Duff: The implementation should define the runtime semantics of these attributes and the RM should get out of the way.

Anthony: Earlier experience with GNAT showed that this attribute produced non-sensible values.

Bob recommends that the Implementation Advice should be eliminated as being overspecified. Mike suggested that implementations document how the attribute is calculated, to which Pascal objects because this is rather intractable in the presence of incremental compilations and smart recompilation. Erhard suggests moving the Advice paragraph in the AI to the Discussion section.

There was no further suggestion to change the conclusions of the AI. Approved with the suggested change regarding the advice paragraph, 8-0-1.

AI-105 extra negative value

The group focused the discussion on the four questions in the initial comment from Dan. The answers to the first three questions are best left implementation-defined (since they are influenced, e.g., by the ISA). The last question did require some analysis that finally concluded that not making it also implementation-defined would cause existing implementations to make significant changes.

The AI was approved as a confirmation, 5-0-2. A letter ballot was requested.

AI-116 Elaboration of task type with no task_definition

After close inspection, the AI didn't seem so frivolous and did represent a binding interpretation. Approved as a binding interpretation, 9-0-0.

AI-117 Calling Conventions

Bob says that this AI will break privacy of the language, which is a serious problem; Erhard counters that this is already the case with the RM rules in an even worse way;

Gary asks for a summary of this AI; Erhard proceeds with three alternatives

- „Inherit“ the calling convention from type where the user defines the conventions per type. This is a convenient way of covering all operations by default; unfortunately it requires the user to look in the private part for private types to know the convention if the user plans to add explicit conventions for type extension. (Conventions are only allowed on full types.)
- „Inherit“ the calling convention from the parent operation where the user defines the convention per operation. Compared to the first alternative, this is somewhat inconvenient for the user because the usual case will be that all operations share the same convention, so that the specs get cluttered up with these pragmas.
- „Don't inherit“ the calling convention and the user must specify conventions of the operations of all extensions, when the 'root'-operation is non-Ada, even though the pragma can only confirm the convention of the 'root'-operation. This is by far the most inconvenient rule for the user, breaks privacy even worse than the first alternative (which at least works fine, as long as the user does not add convention pragmas to subprograms of extensions), but it is the current RM rule.

Bob offers a slight modification of the first alternative, namely, when a convention is applied to a type, it is the default convention of all operations of all types extensions and, for extensions, new operations inherit their default convention from the type and overridden operations inherit from their parent operation. This accommodates the case of mixed conventions for the operations of a type.

After some further discussion of exposing the convention of private types, Erhard makes a proposal:

1. Inherit the convention from the parent subprogram.
2. The subprogram convention defaults to the convention of the type.
3. Declarations of calling conventions of private types are allowed in the visible part.

Ted points out that the last point still doesn't get around the problem of looking into the private parts to follow the derivation tree to find the proper convention.

Several other ideas were explored:

- Make the type view and the full type agree on conventions, which is not upward compatible.
- Make it illegal to change the convention of the type extension, which may prevent flexibility in implementations.

After further discussion, Erhard then adds to the proposal:

4. Remove proposal 3 because, while it is nice to have, it violates rules in Chapter 13 and might be expensive for implementation to change to.
5. Derived types have as default the convention of the parent type (which is the current rule of the language)
6. The convention of a partial view of private types and private extension is the convention of the full type.
7. Change rule 1: Inherit immutable convention from parent subprogram.

In summary, the following set of rules were approved (by 9-0-1):

1. All inherited and overridden subprograms inherit the convention of their parent subprogram.
2. New operations of type extensions have the convention of their type unless a new convention is defined for the operation, if this is supported by an implementation.
3. Derived types have as default the convention of the parent type (which is the current rule of the language).
4. The convention of the partial view of private types and private extension is the convention of the full type.

The AI will be rewritten to reflect this agreement. It might go to a letter ballot, instead of being discussed at the next meeting.

AI-119 Is Normal Termination an "External Interaction"?

Erhard asks for proponents for either side to present their case: Stephen, Laurent and Ted argue against the AI; Bob and Erhard argue for the AI.

It is pointed out that idle tasks/partitions may look like this and they serve a useful purpose, especially in the presence of multiple partition applications for a single processor. Although, in

practice, the idle task will do something and therefore there will be additional activities that will have external interactions.

Brian describes an example of an industrial application where the idle task has this infinite null-loop body and whose priority is raised to the highest priority upon occurrence of a fatal error in order to „halt“ execution and hold the processor.

Erhard calls the vote on whether termination IS an external interaction: 9-2 in favor. Bob will rewrite the AI to say so and to base the decision on a different example, using a while-loop with statically unknown condition, since the „loop null; end loop;“ example is regarded by many as being pathological. The AI will then be the subject of a letter ballot.

AI-125 Order of Size and Small Clauses for Fixed Point

Approved as stated, 6-0-1.

AI-126 Classification of Language-Defined Packages

The last paragraph will be modified to include Preelaborated as well as Pure. The AI will then be put out for letter ballot.

AI-130 Should No_Local_Allocators disallow nested instantiations?

The really thorny issue here is that the locality of allocators in a generic body depends on the locality of the instantiation.

Erhard objects strongly to this AI because he believes that it violates the Generic Contract Model. Bob states that the AARM makes it quite clear that the Generic Contract Model does not apply to link-time rules. Erhard argues that such a position might be acceptable to the extent that units can be dealt with as block boxes and the rules merely check some global property. These specific rules, however, require that the internals of each instantiated body must be (re-)examined and, worse, this reexamination must be done at link time due to the Post-Compilation Rules (due to the use of the Restrictions pragma). Erhard notes that this also applies to No_Task_Hierarchy and No_Nested_Finalization in Annex D. Erhard requested that this AI be tabled while he creates either attachments or a separate AI to support his position. There was no strong objection at the meeting to this action.

Several references to the respective Annexes should be added to the AI.

AI-131 Interface to C -- passing records as parameters of mode 'in'

Erhard reviewed the points for and against the summary of this AI. The brief discussion did not expose any new information. A vote on the AI as written yielded disapproval, 2-3-5. Erhard will bring this AI to WG9 for direction and, depending on the outcome, conduct a letter ballot.

AI-133 Controlling bit ordering

After some examination, it was hard to understand what issue(s) Dan Eilers raises. There are two views:

- _ It is not clear what the issue is, therefore ask Dan to clarify.
- _ Dan is asking if byte flipping needs to be done.

There was a long discussion of big and little endian representations to try to understand what happens when the word size = storage size. In this situation, according to the required interpretation, the bit ordering only describes the representation of the type and implies no byte-flipping transformation.

Bob argues that the RM does provide direction when the word size \geq storage size, namely, an implementation may need to handle changes in representation as an implementation option.

The group decided (6-0-4) that the AI should state:

- _ For the required level of support: the RM supports the first interpretation (by Dan) that there is no byte flipping.
- _ For support beyond the required level, there may be gaps in the representation caused by components straddling address boundaries and resulting byte/bit flipping may have to be done by the implementation.

AI-135 Circular Renamings as Body

Kiyoshi asks about the wording change section for this AI since it is a binding interpretation. Bob is not prepared to create wording on-the-fly but he believes that paragraph 8.5.4(5) would be the target for change.

Approved without change, 6-0-3.

AI-141 Exceptions in Interfaces.C and its Children

There are several typos and editorial changes in the last paragraph of the response:

- _ Change first „exception“ to „operation“.
- _ Add a „with“ in the last sentence.

Approved, 8-1-1.

AI-147 Optimization of Controlled Types

Bob explains that the subtlety of limited vs. non-limited is that with limited you want all the semantics you expected, requiring the prohibition of the mentioned optimizations, and that with non-limited you will accept looser semantics, thereby permitting optimizations (as the RM already does to a limited extent). Bob provided examples of each: when the limited controlled type extension included a semaphore, you don't want to optimize the semaphore operations out of existence, but when the non-limited controlled type extension included reference counts you may wish optimizations to be applied.

The AI was tabled until Erhard can provide new wording for the summary to address the questions raised in the comments.

AI-148 Requeue of protected entry calls

The approved (6-0-3) intent is that the locking is indeed nested, i.e., both locks need to be held for the requeue to be performed. This is a confirmation.

AI-158 T'Class as generic actual type

Nothing has been done with this AI since the last meeting. Bob expected more assistance from Tuck which was not forthcoming. All agree that Tuck needs to be more active in the wording of his proposal to make it succeed.

There are two actions requested by this AI:

- _ Fixing the problem of renaming a primitive operation of the type associated with the class-wide type
- _ Extending the language in an attractive way to enable better interfacing with other OOP languages, like Java.

In the meantime the group did look at three alternatives outside Tuck's proposal:

1. The illegality alternative, i.e., eliminating the problem by making it illegal. In particular there were two choices examined:
 - _ making the renaming generally illegal within the body („assume the worst“ to prevent the contract violation) of a generic unit with a T(<>) type parameter; and making the instantiation by class-wide type illegal for generic specs that include such renaming
 - _ disallowing instantiations with class-wide types

The group found that these choices are too severe to be practical.

2. Creating special case semantics to permit this specific case of renaming of primitive operations in a generic when instantiated with a class-wide type.
3. Adapting the general semantics of renaming to allow the renaming in question (and others). In particular two changes look promising:
 - Gary suggests a rule on instantiation with parameter T'Class that, in the renaming of a primitive operation, the primitive operation of T is used instead of the T'Class.
 - Another change applies to any renaming of primitive operations. The current rule for renaming of primitive operations of tagged type will not produce dispatching. The suggestion is that the renaming now permit dispatching.
 Taken together this appears to do the job.

Erhard recommends that each alternative be written up and attached to the AI for further analysis; this may also stimulate Tuck to improve his proposal. Erhard volunteered to write the alternative generalizing the renaming semantics. Gary and Bob will try the other two alternatives.

AI-159 Shared Variables in Shared_Passive?

Laurent proposes that an implementation have permission to define additional signaling actions, like execution of the RPC.

Atomic might be the mechanism to handle this situation except that Atomic may be rejected. Offer indicates that there is little difference in execution between an entry-less protected object and Atomic especially in the case of simple gets and puts. Shared access can be handled by one or the other. Nothing needs to be done. The counter-argument is that this ignores the need to specify the semantics of shared variables in shared passive partitions (particularly, since the writer of a reusable package need not have partitioning in mind).

Discussion moved to discussing the signaling effect of RPCs. There are two views:

- Bob: The execution of the remote body (whether it was called synchronously or asynchronously) is the logical continuation of the client thread and therefore it is a logical sequential activity. Similar to Tuck's position.
- Ted: The arrival of the RPC at the remote server is a signaling activity.
- There was concern about the cache flushing that must be done at the client to make shared variables work upon RPCs. Bob's interpretation only works if the PCS knows to do the cache flush; Ted's interpretation sees the cache flush as a natural consequence of the signaling. Conversely, since the PCS is supposedly replaceable by COTS implementations, the PCS should really include a cache flush anyway. Mandating the signaling, and hence the flushing, is liable to then cause two cache flushes to be issued, one by the caller of the PCS and one by the PCS. Unfortunately the RM can't discuss cache flushing but instead the RM must use other terms like sequential execution and signaling to do the job.
- Lots of sticky interpretation of signaling vs. sequential execution prevents agreement on this issue. Apparently signaling prevents code movement across the signal point and cause caches to be flushed. Bob suggests that the procedure is executed

sequentially in the thread of the calling partition, which is stronger than signaling; this appears to be a sufficient model for normal RPC. However, it doesn't apply to the asynch. RPC because the body of the asynch. RPC is not sequential with code that follows the asynch. call.

Erhard summarizes the following alternative proposals for the next write-up of this AI:

1. Do nothing, relying on the ramification of the execution of the RPC body being part of the caller's task, and assume that implementation will do the right thing about cache flushing.
2. The body of the procedure associated with the RPC is sequential to the RPC call and the execution of the procedure body of synchronous RPC sequentially precedes the code after the RPC call.
3. An RPC call signals the procedure body and the completion of the procedure associated with a normal synchronous RPC signals the call. Erhard sees a problem that the sites of the call and return will be required to do the cache flush instead of the PCS and, consequently, remote calls to oneself will do two unnecessary cache flushes as well as that all calls might have two instead of one cache flush per signaling at RPC boundaries.
4. The procedure body associated with the RPC is a sequential execution of the thread of the RPC call and part of the calling task and that all the ugly words to handle the asynchronous RPC case will be attached to the asynchronous RPC sections.

The groups supports by a straw-vote, 9-1-3, a write-up along the intent of 4., using the model of 2. for the purposes of 9.10(2), with extra words added that the asynch. RPC does not sequentialize or signal with the caller at the completion of its body.

AI-161 Default-initialized objects

Tabled.

AI-162 Anonymous allocators and tasks/finalization

This AI focuses on the possibilities of finalization of allocated objects of anonymous access types at a more convenient point in the application as an optimization step. In general, for the allocation of anonymous access to types other than controlled and task types, the suggestion by Tuck -- that finalization can be performed at the end of the statement (or outermost expression) where the allocation is made -- appears to work.

The discussion yielded recognition of a related pre-existing problem in the RM.

For controlled and tasks types there is a contradiction between 7.6.1(4) and 3.10.2(13). The example below illustrates the problem:

```

function F (X: access Some_task) return Boolean is ...;
...
procedure P is
...
begin
...
  while G(F(new Some_task, exp)) – (1)
  loop
    ...
  end loop;
end P;

```

By 7.6.1(4)+3.10.2(13), finalization of Some_task is part of the finalization for F. However, if exp raises an exception, F never gets called and hence Some_task never is finalized. Either one of the paragraphs needs to change.

The issue raised by the question should be resolved along the lines of 7.6.1(13), which is already fairly close to what Tucker is proposing.

The example also shows that the finalization rule cannot be applied only to the context of the innermost enclosing expression, but needs to extend to accommodate the use of function results as actual parameters to enclosing calls.

The title of the AI should say „access types“ rather than „allocators“. The AI needs a rather extensive rewrite.

AI-164 Definition of remote access type

Anthony believes that the problem is caused by an oversight of the design team. Erhard worries because he is concerned that there is something that needs to be protected by means of the privateness. Further discussion allays his concerns somewhat.

The code for remote types is replicated on each partition that imports it. The example should be allowed. The only issue seems to be whether to go beyond the requested capability. To be conservative, only the request should be granted.

Since there is the rule prohibiting dereferencing, E.2.2(16), there doesn't seem to be a need to have the 'private' restriction at all; one might as well allow all class-wide limited types.

The decision is to go with simpler wording and wait for someone to object. New wording is to drop „private“ from E.2.2(9).

AI-165 Recursive use of task attributes isn't considered

Erhard suggests that the function/procedures need not be atomic but rather just the actual operations of reading and writing the physical values that need to be atomic. But this raises the issue of race conditions while accessing the task attributes and deadlocking without the addition of locks for each attribute, too. This will be exceedingly expensive to provide.

Erhard does not understand the concern about expense of a lock per attribute. Offer explains that the cost has to do with the creation of locks for each attribute by utilizing OS primitives for locks.

Offer also notes that there will be problems for any operations called out of finalization code that might require a lock on the run-time. If the AI answer allows for an implementation that uses a global RT lock to protect the task attributes, then any such operation initiated by the finalization routines will equally deadlock. In other words, the alternative of using a single lock for the task TCB/attributes will cause any references (during finalization) involving the RTS to result in a deadlock.

Ted describes the mess that the generality of the use of attributes of controlled types creates for implementations.

Offer suggests that an implementation be permitted to restrict the capability of task attributes. This appears hard to state in the RM.

Eliminating task attributes of extensions of controlled types also appears to be too restrictive and again hard to state in the RM.

Erhard and Ted converge on the restriction that none of the operations of the extensions of controlled types should make reference directly or indirectly to any operations of any instantiations of this `Ada.Task_Attributes` package. The users should be so advised.

Alternatively, this could be stated as either

- _ Bounded error
- _ „Works as expected“ (with granularization of locks for each task attribute)
- _ Exception

After asking for recommendations to fix this problem, Erhard proposes that the user is permitted to instantiate this package with an extension of controlled types but that the operations of this instantiation may result in a bounded error, namely deadlock, or work as expected. Unanimously approved.

AI-166 Parameterless_Handler values designating default treatment

What is this AI saying?

Apparently Norm wants to tighten the semantics for the default handler. This is counter to the principle of leaving the semantics vague to permit implementation flexibility, especially on

existing operating systems. Ted complicates the situation by pointing out that an operating system can change the default treatment.

Also, there are some style changes, namely the use of „treatment“ and „handler“ that is causing confusion; Bob indicates that treatment was added to permit the ability of the default to change (by the OS).

Erhard summarizes the alternatives (and prefers no. 1):

1. Current and old handler is null for default treatment; this was modified by Ted to „for any non-user-provided handler/treatment“
2. Current and old handler is not null and callable (by X.all)
3. Current and old handler is not null and may be callable if it is user-defined handler and not callable if it is default handler/treatment

Bob states that the current statement of the RM permits either 1 or 3 as defined by the implementation.

Erhard asks for vote, late in the first day, on Alternative 1: 6-3

On the next day:

Ted convinced himself by the name „Exchange Handler“ that the focus of this operation is handler and not treatments; so that default treatment is expressed by null for the purposes of the Ada application and for later exchange (restoration).

The only remaining issue is the possibility of changing the default treatment from the underlying OS. After some discussion of the terms „current“, „default“ and „current default“, it was decided to stick with „default“ with comments that the OS implementation may change the value of default treatment; „current default“ would only encourage such behavior.

Erhard asks for a revote on Alternative 1: 9-0-0

AI-167 Erroneous scalar Unchecked_Conversion?

There is a dilemma here -- the user can't get to check the validity of the resulting value before the program is „defined“ to be erroneous. Sparse enumerations are a particular source of problems for this AI. One obvious portable solution is to somehow „promptly“ test the validity of the resulting value before the user does anything else to the value. Making this work seems too messy to define. A non-portable solution is to make the situation implementation-defined rather than erroneous. Another non-portable solution is to raise an exception when an invalid value is detected during the conversion. This was rejected during the language design process.

The user can directly handle this problem by doing the unchecked conversion by either:

- Converting to an integer, then using a case statement to check for valid values
- Wrapping the designated result in a record as the sole component. The user can then perform a validity check of the component value. This is due to the fact that there is no component type checking performed when the assignment is to a record type.

The group reached consensus that the only option is to confirm the language on this issue and to expect the user to do the sensible thing to avoid this problem.

AI-168 Aliased objects can have discriminants modified

There are two problems that this example exposes:

- The view conversion example to which Stephen needs to be illegal to prevent the problem.
- There doesn't seem to be a language rule that makes it illegal, but such illegality needs to be part of the solution. It is not a full solution, however, since the discussion showed yet another problem not involving view conversions, as illustrated by the example below.
- The problem with 3.6(11) and the declaration of type A. It appears that the declaration is illegal because A does not have the full view of T and 3.6(11) does not do the right thing.

Bob expanded Stephen's example to illustrate the problem better:

```
package P is
  type T is private;
  a: constant T;
  b: constant T;
private
  type T( X: integer := 0 ) is null record;
type Ptr is access all T(0);
  a: constant T := ( X => 1 );
  b: constant T := ( X => 2 );
end P;
package Q is
  type A is array( 1 .. 10) of aliased P.T;
  type B is array( 1 .. 10) of P.T;
  Foo: A;
end Q;
with Q;
package body P is
  ...
  PT: Ptr := Q.Foo(1)'access;
begin
  Q.Foo := (others => (X =>2));
end P;
```

Bob summarized a solution: for a private type that does not provide a discriminant in the partial view and has a defaulted discriminant in the full declaration, it is illegal to declare a general access type whose designated subtype is constrained. Also, view conversions should be illegal, unless both or none of the types involved are aliased or contain the same aliased components. The answer seems to lie in a modification to 3.10(9).

AI-172 Optional main subprogram?

Priority of the AI is high because it is a surprise to many, including implementors. For example, GNAT does have problems with the current ACVC tests on this issue. GNAT requires the user to designate a main subprogram.

It was suggested that implementations should describe the method for linking a partition without a main program, so that the AVO can exercise that test objective.

The issue is whether an implementation forces the user to create a null main subprogram when no main subprogram is designated. It appears that the RM is specifying the method by saying that the linker command include all the units and saying there is no main subprogram. An implementation should be free to choose its approach. An implementation might require more or less information from the user, such as an Ada-like syntax to the linker that would resemble an Ada main subprogram.

Is the ACVC test LA1001F.ADA test a valid and good test? Most think that the test is invalid. That this test fail at link time is wrong; it should be linked and run successfully (given the „right“ linker command to indicate absence of a main subprogram).

Action: 5 voted for complete freedom; 6 voted for the current wording of summary; all voted that the current ACVC test is wrong.

AI-177 Interfaces.C.Strings.Value with len returning String

After some discussion about the appropriate step to take, it seemed that Ben's initial response contains the best answer to the problem, namely:

Equivalent to `To_Ada(Value(Item, Length) & nul, Trim_Nul => True)`.

Approved with this intent, 4-0-6, letter ballot requested.

AI-179 Finalization and Unchecked_Deallocation

Erhard proposes selecting Alternative 3 (from the AI) and discussion ensued on the possibility. (The word „raises“ in the Summary should be replaced by „propagates“.)

Offer proposed to change „unspecified“ to „implementation defined“. The counter-arguments are that the behavior may well depend on where the exception was raised and for what reason. Also, exceptions in finalization are such bad news that programmers should better see to it that they never arise, rather than attempting fix-ups based on implementation-defined behavior.

Approved for editorial review, 9-0-1.

Note to the user: to gracefully handle the problems of an exception in finalization, attach a when-others exception handler to all finalize subprograms; this permits a local fix-up and no exception propagation will then occur (excluding really bad ones like `Storage_Error`).

AI-180 Elaboration Pragmas & "Mentions"

It was pointed out that if the user wants transitivity, then the user should depend on the `Elaborate_all` pragma and not some contrived interpretation of this pragma. AI approved as stated, 8-0-2.

AI-181 components of `Stream_Element_Array` should be ALIASED

Bob discloses that there is lots of communication that is missing from the AI.

Bob also objects to the use of aliased because there do exist architectures where the stream element size is not equal to storage element size, namely 36-bit word holding six six-bit characters. The use of aliased will make `Stream_Element_Array` completely unnatural for such architectures.

Pascal recommends that an implementation is required to support aliased interpretation of `Stream_Element_Array` when the storage unit size is a multiple of stream element size. For other sizes the implementation has permission to ignore this requirement.

Approved this AI with Pascal's recommendation, 8-1-1. A letter ballot was requested.

AI-182 Extension aggregates with controlled subcomponents

Gary questions why this AI covers both `Initialize` and `Finalize`. Bob responds that handling both operations seemed closely coupled and that he was trying to save AI numbers to avoid the AI-100000 problem...;-).

For clarity sake, Pascal recommends that the „notwithstanding“ phrase be dropped and that the remaining sentence be applied directly to 7.6.1(13).

The group approved the AI with a change according to Pascal's recommendation, 10-0-0.

AI-183 Primitive operations declared before it is known if the type is tagged

The summary can be deduced from the language rules but its consequences ought to be also explicitly stated in the summary to explain the disposition of the two examples of the AI. Additionally, there is an „a“ to be added as a second word of the summary.

Approved with editorial changes, 7-0-0.

AI-184 View conversion to an indefinite subtype

This is tabled until:

- _ more examples can be added to fully illustrate the problem
- _ the definiteness of the types in a derivation tree needs to be more closely looked at and possibly strengthened to become invariant down the derivation chain.

Pascal and Bob will handle this.