Minutes of the 20th ARG Meeting

3-5 October 2003

Sydney, Nova Scotia, Canada

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns, Gary Dismukes, Pascal Leroy, Steve Michell, Erhard Ploedereder, Ed Schonberg, Tucker Taft.

Observers: Christine Agur (Friday morning only).

Meeting Summary

The meeting convened on 3 October 2003 at 09:00 hours and adjourned at 14:15 hours on 5 October 2003. The meeting was held in the boardroom of the National Research Council (NRC), in Sydney, Nova Scotia, Canada. The meeting was hosted by the NRC (Steve Michell). The meeting covered most of the amendment AIs on the agenda and a few normal AIs from the agenda.

AI Summary

The following AIs were approved:

AI-347/01 Title of Annex H (10-0-0)

The following AIs were approved with editorial changes:

AI-218-03/02 Accidental overloading when overriding (10-0-0)
AI-230/07 Generalized use of anonymous types (6-0-3)
AI-252/05 Object.Operation notation (9-0-1)
AI-296/05 Vector and matrix operations (8-0-2)
AI-317/03 Partial parameter lists for formal packages (8-0-2)
AI-348/04 Null procedures (10-0-0)
AI-353/01 New Restrictions identifier No_Synchronous_Control (10-0-0)

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

AI-217-6/04 Limited with clauses (10-0-0)
AI-251/08 Abstract Interfaces to provide multiple inheritance (10-0-0)
AI-254/04 Downward closures for access to subprogram types (8-0-2)
AI-266-2/04 Task termination procedure (10-0-0)
AI-286/04 Assert pragma (10-0-0)
AI-301/05 Missing operations in Ada.Strings.Unbounded (10-0-0)
AI-327/03 Dynamic ceiling priorities (7-0-3)
AI-329/01 pragma No_Return -- procedures that never return (5-1-4)
AI-340/01 Reduce attribute (10-0-0)
AI-351/01 Time operations (10-0-0)
AI-354/01 Group Execution Time Timers (9-0-1)
AI-355/01 Priority Specific Dispatching including Round Robin (7-0-3)

The following AIs were discussed and require rewriting for further discussion or vote:

AI-231/05 Access-to-constant parameters and null-excluding access subtypes
AI-285/04 Support for 16-bit and 32-bit characters
AI-307/05 Execution-Time Clocks
AI-318/01 Returning [limited] objects without copying
AI-326/03 Tagged incomplete types
AI-331/00 10.1.1(19) doesn't handle multiple nesting?

AI-332/02 Resolution of qualified expressions
AI-344/00 Allow nested type extensions
AI-345/01 Protected and task interfaces
AI-356/01 Support for Preemption Level Locking Policy
AI-357/01 Support for Deadlines and Earliest Deadline First Scheduling

The following AIs were voted No Action:

AI-217-1/03 Handling Mutually Dependent Type Definitions that Cross Packages (10-0-0)
AI-217-2/02 Handling mutually recursive types via package "abstracts" (10-0-0)
AI-217-3/01 Handling mutually recursive types via separate incomplete types (10-0-0)
AI-217-4/04 Type stubs with package specifiers (10-0-0)
AI-217-5/02 Type stubs with limited context clauses (10-0-0)
AI-217-7/02 Incomplete type completed in a child (10-0-0)
AI-264/01 Exceptions as Types (10-0-0)
AI-282/01 Ada unit information symbols (8-0-2)
AI-293/01 Built-in hash function (9-0-1)
AI-338/00 Constrained attribute for generic formal private types (10-0-0)
AI-339/01 Add use clauses to protected types (9-0-1)
AI-346/01 Roots of Polynomials (9-0-1)
AI-352/01 ARINC_653_Processes Profile (8-0-2)

Detailed Minutes

Meeting Minutes

Several corrections to the minutes of the 19th ARG meeting had been submitted. Randy indicates that he has already made the needed changes, but had not posted the corrected version in order to avoid confusion. The minutes (with the changes) were approved by acclamation.

Next Meeting

As decided at meeting 19, the next meeting will be hosted by Joyce Tokar in the Phoenix area. The dates of February 27-29, 2004 were selected. Pascal reports that Joyce has been in communication with him about the choice of venue.

The meeting after that will be held after Ada Europe in Palma de Mallorca, Spain; the dates are June 18-20, 2004.

We discuss the October meeting (Randy and Tucker volunteer to host), but decide that it is too far in the future to make a decision.

On Sunday, Pascal queries if it would be possible to have an extra ARG meeting at SIGAda 2003 in San Diego. Doing so would help justify the travel to the WG9 meeting (and to SIGAda) for many attendees, and we have plenty of work to do on critical issues if we are going to stay on schedule. This suggestion gets general support.

Pascal is giving a presentation on Thursday morning, and the WG9 meeting is on Friday morning. Pascal will try to see if he can reschedule the presentation to another time. We tentatively decide on a 2-working day meeting on Thursday afternoon, Friday afternoon, and Saturday, December 11-13, 2003. Pascal will try to move his presentation to ease the scheduling of the meeting.

(After the meeting, Pascal managed to move his presentation to Wednesday, so the final schedule for the 21st meeting is: Thursday, December 11th; Friday afternoon, December 12th; and Saturday morning, December 13th.)

Gary is drafted to find a meeting location and make arrangements.

Motions

The ARG thanks our host, Steve Michell, for the local arrangements for the meeting, with special thanks for the fine dinner on Saturday night. Approved by acclamation.

Old Action Items

The old action items were reviewed. Following is a list of items completed (other items remain open):

Steve Baird:

- AI-251
- AI-291
- AI-348 (split from AI-251)

John Barnes:

- AI-254
- AI-296
- AI-346 (split from AI-296)

Randy Brukardt:

- AI-218-3
- AI-301

Editorial changes only:

- AI-297
- AI-298
- AI-321
- AI-326
- AI-328

Alan Burns:

- AI-266-2
- AI-307 (reopened)
- AI-327

Kiyoshi Ishihata:

- Consult with the Japanese SC22 about the acceptability of AI-285.

Pascal Leroy:

- AI-217-6
- AI-311

Erhard Ploedereder:

- AI-237

Ed Schonberg:

- Create a GNAT implementation report for current version of AI-217-6.

Tucker Taft:

- AI-230
- AI-231
- AI-252
- AI-304
- AI-317

- AI-345 (protected and task interfaces)

Reassignment of Action Items

We decide to look at the list of action items, and reassign some so that some members are not overburdened.

Steve Michell suggests that AI-250 by made No Action in favor of AI-345 (which solves the same problem with fewer complications). This has general approval.

Mike Yoder is retired; so reassign AI-315 to Pascal Leroy.

Mike Yoder and Mike Kamrad have a set of join action items. They had made several proposals based on these, none of which went anywhere. There is no reason to think that any of these others would be different. Make an AC out of these action items (these are all about encouraging more uniform implementations).

Tucker has too many action items. Reassign AI-133 to Pascal Leroy, AI-162 to Steve Baird, AI-188 to Alan Burns, AI-214 to Randy Brukardt, and AI-312 to Gary Dismukes.

New Action Items

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

Steve Baird:

- AI-51
- AI-109
- AI-162
- AI-251
- Create an implementation report for AI-318.

John Barnes:

- AI-254

Randy Brukardt:

- AI-214
- AI-301
- AI-326
- AI-340
- Create an AI to study whether additional predefined units should be preelaboratable (see AI-266-2 discussion).
- Make an AC to encourage more uniformity in implementations. Some suggestions were:
    - External_Tag
    - Storage_IO of tagged types
    - Array indexed by holey enumeration
    - Static elaboration
    - GNAT attributes and pragmas
- Update Janus/Ada implementation report for current version of AI-217-6.

Editorial changes only:

- AI-218-3
- AI-230
- AI-252

- AI-296
- AI-317
- AI-348
- AI-353

Alan Burns:

- AI-188
- AI-266-2
- AI-307
- AI-327
- AI-354
- AI-355
- AI-356
- AI-357

Gary Dismukes:

- AI-312
- AI-331

Bob Duff:

- AI-219
- AI-239
- AI-269
- AI-303
- Be the test creator of last resort.

Pascal Leroy:

- AI-133
- AI-217-6
- AI-285
- AI-315
- Write an article on the Amendment for the Ada Letters.
- Write a report on access subtypes (with Tucker Taft).
- Create an AI on No_Nested_Finalization, extending it to cover types in language-defined packages. (See the discussion of AI-354.)
- Create an AI on "raise with String". (See the discussion of AI-264.)
- Update Apex implementation report for current version of AI-217-6.

Steve Michell:

- AI-148
- AI-250

Ed Schonberg:

- AI-292
- Create an implementation report for AI-318.

Tucker Taft:

- AI-231
- AI-275
- AI-286 (split Assertions_Only into a separate AI).
- AI-288 (split into two AIs: pre/postconditions and invariants).
- AI-290
- AI-295
- AI-318
- AI-329
- AI-332
- AI-344
- AI-345
- Update AdaMagic implementation report for current version of AI-217-6.
- Write a report on access subtypes (with Pascal Leroy).
- Create an AI on the restriction No_Dependence (see the discussion of AI-353).

Items on hold:

- AI-284 (waiting for more keywords to be defined)

Detailed Review

The minutes for the detailed review of AIs are divided into existing amendment AIs and non-amendment AIs. The AIs 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 six votes for, one vote against, and two abstentions.

Detailed Review of Amendment AIs

AI-217-1/03 Handling Mutually Dependent Type Definitions that Cross Packages
AI-217-2/02 Handling mutually recursive types via package "abstracts"
AI-217-3/01 Handling mutually recursive types via separate incomplete types
AI-217-4/04 Type stubs with package specifiers
AI-217-5/02 Type stubs with limited context clauses
AI-217-7/02 Incomplete type completed in a child

After the discussion of **limited with** clauses (see below), we decide to close all of the other alternatives for solving this problem.

No Action (10-0-0) for all 6 alternatives listed above.

AI-217-6/04: Limited with clauses

Pascal explains the differences from the last version.

John notes a typo in the second paragraph of !problem: "among type{s} declared"

Should we discuss "incomplete views", rather than "incomplete types"? The model of the language is inconsistent; this should be fixed. However, that should be done as part of AI-326 (tagged incomplete).

Pascal notes that the biggest changes to the wording are with the interaction of **use** clauses and **limited with**. Items are potentially use visible, but not use-visible. That is that, they still make a unit illegal if there is a conflict. Ed objects — this seems like a new concept that would be hard to implement. Why do we need it? Look at the examples in the AI. Tucker thinks that the problem isn't severe enough to introduce a new concept; you need multiple changes to a unit before there can be a problem. Pascal responds that if we didn't have this rule, the child could see more (in a case like the example) than the parent does. **Limited with** shouldn't never make additional items visible.

Steve Baird suggests that making this **limited with** illegal would be OK; it's not a ripple from some parent. Pascal thinks that this might be useful. Tucker notes that if this had a direct **limited with**, it would be illegal. Why should we worry about allowing it for units with funny renames?

We decide to make the **limited with** illegal if there is use visibility of the package or any of its subpackages.

Tucker writes an example on the whiteboard:

```
with Z.C1;
package Q is
   package RC1 renames Z.C1;  -- Child of Z
end Q;

package P is
   use Q.RC1.NP;     -- NP is a nested package.
end P;

limited with Z; -- Legal?
package P.C2 is
```

This has to be illegal: the parent package P has less visibility than the child package P.C2. This is similar to the previous case — if the **limited with** would be illegal if the parent had a **with** clause, it should be illegal if the parent has a **use** clause.

Tucker notes that this also has to apply to **use type**.

Erhard asks about 10.1.1(26), this seems to introduce an unwanted dependency. No, limited views are elaborated (just like the normal view), but this elaboration does nothing. Steve Baird notes that the package is elaborated twice (first the limited view, then the normal view). But then we have the same phenomenon for private types.

It is suggested that the definition of "limited view" should occur early in 10.1.1 (perhaps in paragraph 12), so that it is defined before it is used.

Tucker notes that it should say "implicit" in 10.1.1(26): "The implicit declaration of the limited view of a library package depends semantically upon the implicit declaration of the limited view of its parent package."

Erhard wonders if this rule is needed at all. Randy and Pascal point out that it is the permission for an implementor to require that the view be available. That's critical to library-based implementations (we have to allow implementations to require that it is in the library before a **limited with** can be processed).

Pascal will update the AI.

Approve intent of AI: 10-0-0.

AI-218-03/02 Accidental overloading when overriding

Randy explains the changes. Overriding indicators are now allowed on bodies. This meant that overriding clauses were no longer required, and they're not useful enough for the added complications, so they were dropped.

Tucker notes an error in the wording: a body isn't an explicit declaration. So 8.3(26) should say "If a subprogram_declaration, ..." Second bullet should say "declaration or body…".

Approve AI with changes: 10-0-0.

AI-00230/07 Generalized use of anonymous types

Tucker notes that he fixed the equality problem by adding type *universal_access* and associated rules.

Should a rename of an anonymous access object (8.5.1(4)) repeat **access constant** when it is needed? Pascal suggests that the existing language is wrong here, but we need to be consistent. Tucker argues that this is different than just **constant**. Steve Baird explains that existing renames do not repeat subtype or object specific details; however this is type-specific information. Doing that implicitly would make the existing renames problems worse.

Tucker notes that **null** is of type *universal_access* now. Is this going to be a problem for some implementations? Several implementers say they have such a type already. The model seems like the most reasonable way to go.

The conversion rules are changed to support *universal_access*. Gary wonders how do we allow **null** to match anything? Because is it universal (see 3.4.1(6)). Does **null** = **null** work? No, **null** is not anonymous access, otherwise the *universal_access* equality would be ambiguous with the normal one for X = **null**. (Too bad, otherwise, you could get X = **null** without **use type**. But that could be incompatible if a user-defined equality is defined, so we can't do that.)

Tucker tries to explain the reasons for 13.11(25).

```
type LT (D : access XT := new XT) is limited ...

new LT (D => new XT); -- An allocator
new LT; -- Another allocator
```

Erhard asks why this doesn't have the accessibility level of XT? That would be a big incompatibility:

```
Loc : aliased XT;
Loc2 : LT (Loc'Access);
```

is legal and safe in Ada 95.

Returning to the original question. We don't say much about it the RM

```
type LT (D : access XT) is limited
   record
      Z : Lim_Type (D => new XT);
   end record;
```

These all have the same accessibility, thus the same pool makes sense.

Approve AI with changes: 6-0-3.

AI-231/05 Access-to-constant parameters and null-excluding access subtypes

Tucker explains that a major change here was to call these "null exclusion"s instead of "not-null constraint"s. This avoids double constraining and simplifies the model.

John notes that in the syntax, general_access_modifier should be in square brackets.

Tucker comments that the wording change in 4.6(49) should really be in 4.6(51).

Static matching must include null exclusion; the AI needs to include that.

Pascal notes that we need some rule for formal / actual generic matching for formal access type.

After much discussion, we decide this is a subtype property, and thus it should inherit it from the actual (like a range constraint). Otherwise, we would need two Unchecked_Deallocations!

Steve Baird points out that Unchecked_Deallocation for a null-excluding subtype would always raise Constraint_Error (as it is defined to return **null** in the parameter). Thus it is not useful anyway on such types.

That seems to argue for the formal / actual generic matching. We could have a "swap-and-free" version of Unchecked_Deallocation. Straw poll: Do we have null exclusion on generic formals: for: 3; against:5; abstain: 2. Examples would be useful to better understand the implications (this could lead to a clearer vote).

Can you say "**not null** T" in a subprogram formal parameter? No. Perhaps we should allow that; the reason Ada doesn't allow it is for elaboration reasons, which don't apply here.

For subtype conformance, null exclusion needs to match.

We decide not to vote on intent, because a number of new issues have surfaced which require some study. Instead, we vote to continue work on the AI: 10-0-0. Tucker will update the AI.

AI-251/08 Abstract Interfaces to provide multiple inheritance

Steve Baird outlines the changes. He notes that Tucker has more recently noticed that the syntax is inconsistent. The syntax in 3.4 is correct, 7.3 and 12.5.1 should be like it.

```
[abstract] new ancestor_subtype_indication [and interface_list]
            with private;
```

Ed asks about the implementation model; was it 'original'. Steve Baird explains that it is question of precomputing vs. overhead at runtime.

Move the wording before the !discussion.

This is inheriting of a contract, not any implementation.

John would like some baby examples in the proposal; just so the flavor of the proposal can be seen easily.

Erhard wonders about the syntax for interface composition. It requires a record extension part. No, that's not how you make a new interface (note 9 is wrong).

```
type T1 is interface with A and B;  -- A new interface.

type T2 new T3 and T1 and A with null record; -- A new derived type.
```

Tucker: "An interface type shall have no components" is not a legality rule; it is static semantics and should read "An interface type has no components."

8.3(12.4): "shall" -> "is".

Erhard asks if we can have different defaults in two inherited routines with the same name. No, they must be fully conformant. Otherwise, they both have to be overridden (which provides default expressions).

"do not cause [any of] them to be overridden" (also 8.3(12.4)).

Tucker would prefer to change the 8.3 wording to try to be less 'mysterious'. He claims that you only care about the result (which routine is defined). Pascal claims that the existing wording of the standard requires this. Inheritance is defined in 3.4; 8.3 tells us what is visible.

Tucker starts reading the existing RM. Seems confusing.

Tucker thinks that the source of the operations doesn't matter, only concrete has precedence over null which has precedence over abstract, no matter where they come from (i.e., even if a null procedure comes from an interface). Steve Baird thinks that the source matters; non-interface should trump. Ed mentions that Java often uses 'adaptors' to provide null interfaces. But it works in the other order.

Steve Baird thinks it is weird to fill in obligations with an interface. Pascal says that would mean that inheriting from interface is more than a contract.

Tucker thinks that changing from abstract to interface would change the semantics if the source matters.

Ed thinks that perhaps we should drop null procedures from interfaces. Tucker points out that a generic unit can assume that anything that has an interface with null procedure would have non-abstract implementation — so it is always safe to call. That property would not be true if you don’t have Tucker’s interpretation. A null operation must not revert to abstract — otherwise you cannot write useful generics.

This is compelling; Steve Baird gives in and will rewrite this that way. Subprograms only need to be fully conformant if they are the same kind.

Get rid of “unspecified” from the last paragraph of the visibility wording.

The wording still is clear as mud. What is being done here? We look at inheritance as it is (T3 inherits from T1 and T2, T2 from T1):

T1 Foo-1

T2 Foo-1a (implicit); Foo-2 (explicit)

T3 Foo 1b, 1ab, 2b -- Three Foos (note: 2 may not be fully conformant).

8.3(12.2) seems unnecessary, because this is the same rule as in the existing language. Steve Baird says that this is used to make Foo 1ab hidden. People object: we’re not using the interface here. Erhard and Tucker read the RM, and point out that Foo 1ab is not inherited — it is never declared, and thus never visible.

It needs to talk in terms of declarations, not inherited subprograms.

Gary would like to know why the third rule in 4.6 exists. Steve Baird replies that we want to allow any membership test that might return True.

Tucker is confused about 4.6. He thinks that the wording is in the wrong places. We finally convince him that the wording is organized correctly.

Gary reasks his question: isn’t it statically known if there is an interface. Not for class-wide types, some child type could implement that interface.

So, the only big change to this AI is that null procedures win when inheriting.

Approve intent of AI: 10-0-0.

Steve Baird will take another crack at this AI.

AI-252/05 Object.Operation notation

Tucker explains the changes to the AI. They're all pretty minor.

Pascal wonders how this works for a case where a component and subprogram conflict. Tucker notes that this is a name resolution rule, and the presence of a component means the subprogram isn’t considered.

Gary comments that there is a Beaujolais-like effect: adding or removing a component could change the meaning of the program. But this seems unlikely, and there are already similar cases in the language.

Randy says that the Intrinsic convention text should only be in 6.3.1, not in 4.1.3. Pascal agrees. So delete the “, and has convention Intrinsic”.

Erhard still isn’t sure that he understands the model. He writes the following example:

```
package P is
   type T is tagged ...
end P;

package P2 is
   type NT is new T with ...;
   type T2 is tagged ...
   procedure Pro1 (X : P.T; Y : T2);
   procedure Pro2 (X : P.T'Class; Y : T2);
end P2;

O : P2.NT;
O.Pro1 (...); -- Illegal. (O is not the right type.)
O.Pro2 (...); -- Legal (O matches the classwide.)
```

Randy suggests removing the note about the unification for task and protected operations being irrelevant. It does unify that, just not completely. The paragraph (second of discussion) should be revised. Tucker would also like to add a paragraph based on the discussion of class-wide operations from the minutes of the last meeting.

Gary wonders if these calls resolve a renaming. A renaming is a view of a subprogram, not a subprogram. Gary suggests that you might add renaming of a class-wide in each package. That would be ambiguous here.

It seems very weird not to include renames here. Add "view of a subprogram".

Someone notes that you can use operator symbols here. Obj."="(X). That's ugly, but there's no good reason to restrict it.

Approve AI with changes: 9-0-1.

AI-254/04 Downward closures for access to subprogram types

Should we allow these as access discriminants, and components, etc.? (This would normalize this with AI-230.) If so, in these cases we would have well-defined accessibility-levels (these would be "normal" access-to-subprogram types); we wouldn't use the infinite accessibility level in those cases.

Erhard notes a couple of spelling errors. "specifiying"; more.

The last sentence and note in 6.3.1(13) are unnecessary: you can't name the anonymous access-to-subprogram to change the convention. Also remove the phrase "(commonly called downward closures)" from the wording; change the title of the AI to avoid "downward closures". John will revise this based on AI-230 & AI-231 as they are currently, and will make these usable everywhere.

Approve intent of AI: 8-0-2.

AI-264/01 Exceptions as Types

Gary comments that the **raise with** message syntax suggested in e-mail should be supported. But there is not much support for the full generality here.

No action (10-0-0).

We take a straw vote on adding **raise with** message syntax. (7-1-2). Pascal will create a new AI for that.

AI-266-02/04 Task termination procedure

Alan reminds us of the suggestions from the last meeting.

Randy notes that there are typos in the last line of Set_Handler.

Why does Set_Handler set any handlers, but Get_Handler does not return them? Get_Handler should take a task id. Steve Michell asks that the Task_Id parameter be last, and with a default value of Current_Id. Tucker thinks that the default isn't useful; you would rarely want to find out your own handler. He would rather leave these alone. Steve Baird says that you could use this to fake your own death. Steve Michell says that it is common for tasks to set up their own handler. But that leaves no handler during activation of the task. Tucker's opinion carries.

Tucker notes that Exception_Occurrence should be initialized to Null_Occurrence.

How do you figure out what the old default handler was? Add Old_Default_Handler to Set_Dependents_Default_Handler.

The use clauses should be removed from the package specification.

Should say that calling these with Null_Id raises an exception. This package should be marked preelaborable. Can it be? Ada.Exceptions is not preelaborable. Why not? There doesn't seem to be any problem. [Editor's note: This wasn't possible in Ada 95, because it has objects of a private type (see 10.2.1(9)). Ada 200Y has pragma Preelaborable_Initialization, which can be used to work around this problem. So, this pragma ought to be put on Exception_Id and Exception_Occurrence, and then the package can be preelaborable. AI-161 looked at existing preelaborable packages, but should also have looked at packages that aren't preelabotable, in case that was caused by this problem, as for Ada.Exceptions.]

General note: We need an AI to see if any of these (Task_ID, Exceptions, etc.) should be preelaborable. There was an old AI 126, but it didn't check carefully if other packages should be made preelaborable. Similarly, AI-161 made it possible for more packages to be preelaborable. Randy will create an AI on this topic.

The model is that the handler is identified when the task terminates; the local handler if any, the master's default handler if any, and so on. That could cause a race condition, but only if termination handler is changed frequently, which is not the intended mode of operation.

Tucker wants to clarify the model.

Task: Handler for dependents; handler for self.

Child task: Handler for dependents; handler for self.

If you call Get_Handler for child task, do you get the default for your parent if there is no handler for self?

Alan explains that you need to be able to get "your" handler for chaining purposes.

But the other possibility is also useful. So we need two Gets. Add a Get_Inherited_Handler.

Wording is needed for the AI.

Approve intent of AI: 10-0-0.

Alan will update the AI again.

AI-282/01 Ada unit information symbols

This issue doesn't look important enough to spend some of our limited remaining time.

No action: 8-0-2.

AI-285/04 Support for 16-bit and 32-bit characters

Kiyoshi reported that Japan won't allow reference to Unicode in the standard. Steve Michell notes that other countries have taken the same position. However, 10646 says very little about upper/lower case, character classification, etc. We need this to properly support strings and identifiers. The new 10646 (in process) will adopt much of the Unicode material, and would help.

Someone asks why we don't just support Wide_Wide_Characters and Wide_Wide_Strings without semantics? This doesn’t buy much — the big win is identifiers.

Our options: just drop improved character support (and see if SC22 objects), or do it with Unicode (and see if SC22 objects).

We decided to go ahead using Unicode, and hope that the new 10646 gets approved in time that we can use it instead. Pascal will produce a new version.

AI-286/04 Assert pragma

Tucker explains the changes. He made other assertion policies implementation-defined to simplify discussion.

Tucker also added an Assertion_Only pragma, so that objects and functions can be defined specifically for assertions. Steve Michell wonders if this could actually be useful; you couldn’t assign to such an object.

```
procedure P is
   X : Integer;
   Y : Integer;
   Z : Integer;
   pragma Assertions_Only (Y, Z);
begin
   ... -- Sets Y, Z – but that’s illegal!
   pragma Assert (X = Y-Z);
```

Tucker suggests instead

```
function Init return Boolean is
begin
   Y := 3;
   Z := 4;
   return True;
end Init;

pragma Assertions_Only (Init);

pragma Assert (Init);
```

Randy: Why is unreachable code analysis not good enough for this? Pascal: Why are we defining a heavy feature for an optimization issue?

Erhard speaks in favor of the pragma; the user would be able to ignore the assert code when reading the program. Pascal asks why comments are not good enough for that.

Steve Michell says that this is useful idea, but it has to be designed carefully.

Randy notes that we suggest not using side-effects in functions, yet this does. Tucker says that we allow side-effects on Assertions_Only entities.

Pascal says that this check would have to be done on all object references. Steve Baird claims that a compiler would not be able to eliminate a variable in a package specification with these rules; but that is what the user would expect. Tucker thinks it could be made to do so.

Alan asks if an Assert can raise an exception other than Assertion_Error. Yes, under the standard policies (it’s too expensive to always raise Assertion_Error, but an implementation-defined policy could do that).

Erhard suggests automatic ghost variables. Yikes! A block is suggested instead.

```
declare
   pragma Assertions_Only;
   ----
   ----  (X := Y);
begin
   ----
   ----
end;
```

Pascal thinks that trying to add syntax here will kill this proposal.

Ed notes that most existing implementations support pragma Assert; how many users complain about this not being expressive enough? Alan says that the HRG is happy with the rest of the proposal, but not necessarily with Assertions_Only.

Tucker says that he sees this problem in his programs.

Tucker suggests splitting pragma Assertions_Only into a separate AI. That gets general agreement. We turn to the rest of the AI.

There is no wording, but the proposal is written as wording. However, we need to say where it goes in the standard. The paragraph starting with “Other implementation-defined policies” is an AARM note.

When the policy is Ignore, the pragma arguments are still resolved. They’re just not evaluated. We want the contents of the pragma to be legal for all policies.

Pascal says that that the question (“should we add”) should be answered yes; this text should be added.

The presence of a pragma Assert should create an implicit semantic dependence on Ada.Assertions.

Steve Baird asks where the Assertion_Policy is in effect. A compilation unit. This is similar to Unsuppress. Defining this properly is too hard to do during the meeting.

Approve intent of AI: 10-0-0

Tucker will update the AI, splitting out Assertions_Only into a new AI, adding wording to define where Assertion_Policy is in effect, and the other changes suggested.

AI-293/00 Built-in hash function

This issue seems too complex and not important enough for our limited remaining time.

No action: 9-0-1.

AI-296/05 Vector and matrix operations

John explains the changes to the AI. He concluded that LU didn’t buy enough, so he took it out of the package. Similarly, he thinks that too much was put in for eigenvalues. Some of them are hard to calculate; we don’t want to require anything beyond what is relatively well-understood.

The child names conflict. We try to figure out alternative names. Why have these in child packages at all, there are only a few routines? So, we move the eigenvalue routines into the main packages, with the names Eigenvalues and Eigensystem.

Should we check for symmetric or hermitian matrices? People think that is too hard. Alternative suggested was to just pass a parameter to say if it was upper- or lower-triangular. Then there would be no check at all. Perhaps there should be predicates so the user can pre-check that. John Barnes complains that such predicates have problems with accuracy. Tucker says that it could be unspecified what happens. John thinks that it would be better to leave it alone. Tucker comments that a predicate could have an error bounds. He’d prefer that you avoid the check. Perhaps, just say use the upper part. John, after reflection, decides to leave it as it is.

When there is a question, stay with accuracy. This is not for very large problems; large problems will probably use BLAS or something like it.

Erhard looks on the net and finds a BLAS binding for Ada. That's too large, and too thin for the standard.

Approve AI with changes: 8-0-2.

AI-301/05 Missing operations in Ada.Strings.Unbounded

Editorial: Fix spelling of Unbounded in !summary. "[Some] additional...".

Change title to "Operations on language-defined string types".

The equivalence should be either written in words, or at least written in such a way that the code would work — this code would always raise Storage_Error.

Gary says: In comments for Index functions, spell "propogated" correctly.

Tucker would like a one-line description of what is new in the Index functions. "Additional Index functions are added to Ada.Strings.Fixed, with From parameters so that partial string searches can be made".

Pascal cannot believe that we should have the "+". It is "disgggusting". Tucker says he thinks it is beautiful.

Steve Baird says that "+" should be a renaming of "To_Unbounded_String".

John speaks up in support; he used this is his book. Steve Michell thinks it would be confusing with concatenation — but that is another language.

Straw vote for the "+" operator: 3-4-3.

So kill the "+" operators.

Unbounded_Slice sounds funny; Ed suggests Slice_of_Unbounded; Tucker suggests Slice_Unbounded. We could use a child, but it would look like a wart.

Straw poll: 6 Unbounded_Slice; 2 Slice_Unbounded; 2 Abstain.

!problem last paragraph: "An child" => "A child". "would be valuable" => "are valuable".

Erhard would argue against the function form of Get_Line, because there is no such form in Text_IO. Steve Baird says this is the most annoying thing for beginners in Ada. Several others also speak in support. It would be better to add it to Text_IO rather than remove it here. (Aside: would that be incompatible?).

Pascal thinks we should add this to Text_IO, and we could do it in this AI.

Straw vote: 5-0-5. We discuss putting this into a separate AI. But we can't agree on where to split the AI (all of the IO subprograms together, or just Text_IO by itself?). Put it into the same AI.

Approve intent of AI: 10-0-0.

Randy will make the changes, and provide wording.

AI-307/05 Execution-Time Clocks

Alan explains the problem with the previously approved AI. You couldn't write a single task that could handle a series of events. So the package is inverted, and we now pass in a protected procedure handler.

Tucker wonders what was lost by this change? It's no longer possible to do a select with abort based on execution time (that is, a deadline). Erhard notes that it could be programmed with a PO. But you can no longer do it directly.

This looks more like an interrupt handler; perhaps the operation set should be made more similar?

Alan will update the AI so that it can be discussed at the next meeting.

AI-317/03 Partial parameter lists for formal packages

Tucker says that this AI is complete except for the Formal.Formal.xxxx problem. He doesn't recall exactly what this problem is.

Erhard says it would happen in something like the following: build a hash table of buckets, then build another container holding it, and we want to get back to the original element type. He writes an example:

```
type Elem_Type;
package Bucket is new Container (Elem_Type);
package Hashtable is new Table (Bucket, ...);

generic
   type Elem_Type is private;
package Container is
   ...

generic
   type E is private;
   with package Bucket is new Container (E);
package Hashtable is
   ...

generic
   with package Hashtable is new Table (<>);
   with package XYZ is new Blah (Hashtable.Bucket.Elem_Type);
           -- Illegal; but Hashtable.E is OK.
```

But there is a workaround (using Hashtable.E). Is there a case where you couldn't get at the type?

The reason this isn't possible is that it isn't clear what the operations of the type should be. If you have a <>, you only have the operations of the formal available. However, if you can see the actual, we don't allow accessing the formal, because otherwise there would be two sets of operations.

Erhard will try to create an example of the problem where it can't be worked around. He does this while the discussion continues.

Tucker explains the other changes.

Pascal notes a typo in !discussion "canÕt" should be "cannot". Ed notes an extra word in: !proposal, penultimate paragraph "prefix [may] only when the".

Erhard hasn't found an example; we'll ignore Erhard's problem for this AI. Therefore, remove the [Note: …] from the AI.

Approve AI with changes: 8-0-2.

AI-318/01 Returning [limited] objects without copying

Tucker explains the AI. A function could be used in the same ways as limited aggregates.

Pascal notes that he prefers a previous proposal; Randy concurs. The two competing suggestions are:

```
function Fun (...) return Lim_Type is
   Result: return Lim_Type [:= expr;]
begin

   Result.X := ...;

   return Result;
end Fun;
```

or

```
function Fun (...) return Lim_Type is
begin
   return Result : Lim_Type [:= Expr] do
       Result.X := ...;
   end Result;
end Fun;
```

The ‘return object’ *looks* simpler, but the semantic rules are ugly. If you have multiple or dynamic constraints, you can put the item into a block. But then you need a rule that it is returned before the end of the block. That happens for free in the second proposal.

The group agrees that the second syntax is preferable.

Tucker and Steve Baird engage in a short discussion of implementation issues. A storage descriptor would be passed to the function (to allocate if the size is not known; to provide the space if the size is known, plus master information).

Steve Michell asks if these calls can be ‘recursive’; can the default expression be a similar function? Sure, you’ve have to pass the storage descriptor onwards.

Can the ‘do list’ contain a return statement? Or just a regular return without an expression? Certainly, would not allow expressions. Tucker would prefer to just disallow return statements.

Steve Baird wonders what happens if an exception occurs in the do block. Who finalizes it? Tucker says that the object belongs to the caller, so it would have to be done there. Steve thinks that that could be difficult in their implementation.

Keep the AI alive: 6-1-3.

Steve Baird and Ed will do an analysis of implementation impact; Tucker will write wording.

AI-326/03 Tagged incomplete types

Randy explains the change, which is to allow dereferences of tagged incomplete objects if they are used as the actual parameter in a non-dispatching call. Ed asks how it would be used; he wonders how you get a value of a tagged incomplete type. Randy replies that it is most useful with **limited with**, but also could be useful for children if the tagged incomplete type was deferred to the body.

Tucker writes an example:

```
limited with Depts;
package Emps is
    type Employee is private;
    procedure Assign (E : Employee; D : Depts.Department);
    type Dept_Ptr is access Depts.Department;
end Emps;

with Emps;
procedure Recruit (D : Emps.Dept_Ptr; E : Emps.Employee) is
```

```
        begin
            Emps.Assign (E, D.all);
        end Recruit;
```

The access-to-incomplete value is passed from a caller (who presumably has visibility on the full type). Why should Recruit be required to **with** a package (Depts) that it is not using and doesn't necessarily know about? From the perspective of Recruit, Dept_Ptr is just a handle.

Is this a new capability? Not really, since tagged types are by-reference types, there is little difference between an **access** parameter with any designated type and an **in** parameter of an incomplete tagged type.

```
        type T;
        procedure Print (X : access T); -- Already legal
                                        -- and possibly dispatching.

        type T is tagged;
        procedure Print (X : T); -- Would be legal and has the
                                 -- same problems as above.
```

We need to change the existing rules to disallow dispatching calls in both of these cases. Once we've done that, we can simply allow dereference as an actual parameter, and let the anti-dispatching rule eliminate any problems.

Whatever rules are adopted should be as similar as possible to access parameters.

There is an overlap in the wording with AI-217; that wording should all be in AI-326.

Randy will update the wording in consultation with Pascal.

AI-327/03 Dynamic ceiling priorities

We discuss whether the object attribute originally suggested is preferable, or whether we like the way the AI is written. If you use the procedure, then you can pass it to generics, etc. We take a straw poll: Attribute: 6; procedure: 3; abstain: 1

Alan then notes that you do need to keep both values — the current one, and the one that will be set.

We discuss the bounded error case; the "both or neither" is bizarre. Randy notes that the fact that the RM currently uses it isn't a good excuse — two wrongs don't make a right. It's confusing. Suggest making it a bullet list. Fix both places in the standard or preferably fix D.5(11), and refer to this section from the other.

It isn't allowed to look at the priority in the barrier. Using the priority in the barrier can lead to incorrect programming. It appears that you could wait for the priority to increase, but that cannot work. Several people say that this special rule seems unnecessary. Let people shoot themselves in the foot. Alan thinks there already is such an attribute with a similar rule, but no one is able to find it.

We can't make this illegal if Ceiling_Locking is not specified; Ceiling_Locking is partition-wide and you don't necessarily know whether it applies to the current unit. Say that the meaning of setting the priority is implementation-defined if Ceiling_Locking isn't specified.

To summarize: The barrier can use 'Priority.; reword the bounded error; use the object attribute. Alan will update the AI.

Approve intent of AI: 7-0-3.

AI-329/01 pragma No_Return -- procedures that never return

Tucker explains the proposal.

The AI should include that Ada.Exceptions.Raise_Exception has this pragma, it will always raise an exception. (That will require a change to Raise_Exception.) The pragma would not apply to Reraise_Occurrence.

Pascal says that he has no strong argument against this pragma, but he finds it disgusting. From an Ada perspective, the user doesn't seem to benefit. Tucker says that it is useful for static analysis.

This pragma comes from GNAT, and is in Sofcheck's compilers as well. Pascal says that it's just one property out of many.

This property applies to few subprograms, which typically are called frequently.

Where should this go? Could go with Inline, or with Ada.Exceptions.

Approve intent of AI: 5-1-4. Pascal doesn't think this is useful enough.

Tucker will update the AI.

AI-338/00 Constrained attribute for generic formal private types

The proposed definition is confusing — 'Constrained would not mean "is constrained". The examples all require various non-portable fiddling. It doesn't seem enough benefit for the effort — it is rare to make something as general as Sequential_IO.

No action: 10-0-0.

AI-339/01 Add use clauses to protected types

Tucker thinks the discussion undermines the AI. Randy says he wrote this up, but he's not in favor of it.

No Action: 10-0-0.

AI-340/01 Reduce attribute

Erhard thinks that the AI looks messy. Pascal concurs, saying that the discussion isn't the best. The problem is intermixing signed and unsigned math, and that should be emphasized. Erhard wonders if there is a better name for the attribute. Tucker says he looked for alternatives and didn't find any. Reduce seems like the best choice; maybe an example showing a large number being fit into a small modular number would show that better. Pascal notes that in the context of a generic formal modular type, it is very hard to get this right.

Tucker wonders if this should be defined for all integer types. That would provide a conversion to go back to a signed value. But that seems harder to define. Steve Baird shows us what it should do on his fingers, but the minute-taker isn't able to record that here. We argue about how hard it is. Tucker suggests the definition would be similar to S'Remainder (for floats). If V < Int'Last then V else V + T'Modulus.

More ideas are tossed out. T'Mod gets support from Erhard. Pascal agrees. Steve Baird wonders why we'd only do one such operator. If we have that, shouldn't we have T'Abs, too? T'Modulo would also work. Pascal thinks that would be confusing.

Straw poll: 'Mod 4; 'Modulo 5; 1 abstain. Ed speaks against 'Modulo, thinks there would be too many similar things. Steve Michell concurs.

It's getting too close to dinner. Lots of screwy ideas are tossed out. (This discussion occurred at the end of the Friday session.)

We decide to retake the straw poll: 'Mod 7; 'Modulo; 2 abstain.

Change the name of the attribute to 'Mod. Improve the discussion: show intermixing of types; and show how it could be used in a generic (which is much harder). Randy will update the AI.

Approve intent of AI: 10-0-0.

AI-344/00 Allow nested type extensions

Erhard wonders about accessing local objects outside of their type definition. Tucker claims that the checks he proposes would prevent that.

Randy says that it would help with the use of containers. But he doesn't particularly want to implement it.

We're undecided as how to proceed. We take a vote: No action: 4; keeping it alive 4; abstain 2. This is not very conclusive (or promising).

Tucker will try a write-up, and we'll talk about it next time.

AI-345/01 Protected and task interfaces

Tucker explains the basic idea. It's hard to inherit protected code; moreover, you want all of the monitor (synchronization) code in a single place. But an interface can be shared and inherited safely.

Are these types tagged? No, but they still have a class. This is not derivation; it's composition.

Steve Michell finds it valuable that this works for tasks as well as protected objects.

Tucker claims that there isn't much work in the task runtime, you would need a look up table to find the entry number or address, but otherwise it would work the same as current tasks and protected objects.

Alan notes that the real-time workshop was very positive for this proposal.

Pascal notes that this looks like a big change.

Steve Baird wonders what these entities are. They are abstract task types and abstract protected types.

```
X : T'Class; -- A parameter

abort X;
X.E(...);
```

There is a taglike item in the object; you have to tell what specific type the object is.

Pascal comments that this looks very much like a tagged type. Perhaps we should just call it that.

Tucker is proposing that the specific type for a protected/task interface cannot be used as a formal type. (Class-wide is OK). Thus, we don't get dispatching routines (which would require additional runtime support).

These are separate entities; protected interface; task interface; and tagged interface will never be mixable. That's unfortunate — instead of unifying OO, we end up with even more features.

Alan notes that they had wanted access to protected entry and access to task entry. But they don't need it if they have this. Pascal says that would be a much smaller change.

Gary asks Alan if this would be useful or a nice to have. Alan says that they find that there are important problems that can be solved with this that can't be solved other ways. So it is more than a nice-to-have (but not critical).

Steve Michell suggests that this could be used to select the appropriate implementation at runtime.

Keep the AI alive: 5-1-4.

Tucker suggests determining the wording first before trying to decide on whether to have it at all.

Erhard wonders why multiple inheritance. A task is a thread of control; if it needs to support multiple interfaces, it cannot use composition, as regular objects can.

```
protected type PT with PI1 and PI2 is...

protected type PT is new PI1 and PI2 with
    procedure Set (X : Integer);
```

We turn to the syntax of the feature:

```
protected [type] interface [type?] PI3 is [new PI4 and PI5 with] ...

protected [type?] interface [type?] PI3 [with PI4 and PI5] [is ...];
```

Compare to AI-251's syntax:

```
type NT is new T1 and I1 and I2 with ...

type NI is interface [with I1 and I2];
```

Tucker suggests alternatives to the AI-251 syntax:

```
interface type NI [is new I1 and I2];

[type] interface NI [is new I1 and I2];

tagged interface NI [is new I1 and I2];

interface NI [with I1 and I2];
```

Some people express concern about syntax that doesn't include **type** for a type declaration. Tucker claims that the current syntax is a derivation, and doesn't reflect that. Randy says that it isn't a derived type, and it shouldn't look like one.

```
interface type NI [is new I1 and I2];

type NI is interface [I1 and I2];
```

Get rid of **with** from the AI-251 syntax.

Pascal opines that **task type** in Ada 83 was a wart; Ada 95 added another such wart. Let's not do more. Randy suggests that this prefix notation is a 'real-time' thing; we don't want to bring it to the sequential world.

Erhard thinks the productions are weird. They come straight from RM grammar.

We decide not to decide on syntax; Tucker will suggest something when he updates the AI.

AI-346/01 Roots of Polynomials

John explains that he was asked to look at other possible numeric packages. He did so, and only roots of polynomials make sense as something that could be in the standard.

Should this be a child? Perhaps it should be a separate unit, with its own generic parameters, a formal floating-point type and an instantiation of Generic_Real_Array or a formal array type.

Pascal wonders if this is of any practical value. No one seems to be able to think of any. John says he's not in favor of adding this to the standard.

No action: 9-0-1.

AI-347/01 Title of Annex H

Why is Ravenscar in here? Some members think it doesn't belong here; it should be in AI-249. Alan explains that Ravenscar is in Annex D; so it makes sense to do this separately from AI-249. Why make more work for everyone by having a separate AI?

Approve AI: 10-0-0.

AI-348/04 Null procedures

Steve Baird describes the changes.

Randy wonders why we need to specify the convention of the actual to a formal subprogram. You could rename the formal in the specification and use it outside of the generic. In that case, you need to know the convention.

12.6(16) is a note, the text should be less formal. "A null procedure as a subprogram default has convention Intrinsic (see 6.3.1)."

Approve AI with changes: 10-0-0.

AI-351/01 Time operations

Tucker would prefer a different name than Ada.Calendar.Operations (but he doesn't suggest an alternative).

Pascal notes that Seconds should include 60, so that we can handle leap seconds. Tucker finds that a problem that there is much code that makes the assumption that there are always 60 seconds in a minute. Pascal says that time servers will give you leap seconds, what do you do with them? Tucker thinks that we should just cover it up. But then Day_Duration would need to be changed (to allow 86401 seconds in a day). That could be a compatibility problem.

There can only be one leap second per day. That is defined by an international time standard; we can count on it.

The fix would be to have the clock "stop" during a leap second. But differences should still work and give the correct number of seconds. (Otherwise, program delays could run a second too long, which could be catastrophic). This sounds ugly. John suggests finding out the number of leap seconds between two times.

Tucker's position is that you never want to get a leap second from Ada.Calendar.Split. 11:59:59|11:59:60|12:00:00 -- the middle result is never given. Differences work properly. Split "stopped".

Add a second Split that returns a Boolean for Leap_Seconds. Time_Of should have a default parameter for Leap_Second := False. But we have to be careful: we don't want to force connecting Ada to a time base.

Add an implementation permission that if we don't have leap second info, we don't return it.

Steve Baird would like to change Days_in_Week_Type to Weekday. Randy: what about Sunday? Weekend_Day? ☺ Ed says that GNAT calls it Day_Name. That gets some support. Should it start with Monday? Yes, ISO standard says weeks start with Monday.

Pascal notes that the parameter name and the type have the same name: Day_of_Week (Time : Time) — better fix the parameter name.

Pascal gives the editor permission to append his private mail on this topic to the AI, so that we have the ISO standard references.

Moving on to the day arithmetic. Steve Baird points out that Seconds can be zero if Left < Right. So the wording should say both can be non-positive. Tucker would prefer saying Seconds <= 0.0 and Days <= 0, rather than "non-positive".

John's leap second count function reappears. Several people agree with that. The function would be:

```
function Number_of_Leap_Seconds (Left, Right : Time) return Natural;
```

Pascal summarizes: Leap seconds are in general are supported; implementation permission to not support them. Others would prefer that it is just implementation advice. This function would return 0 if you don’t support leap seconds. Tucker would prefer implementation advice that you do support leap seconds if available. Pascal decides to agree.

A day is 86400.0 seconds. Add an overload of Difference that returns seconds and leap seconds. That would also be John’s function; we don’t need a separate one. “Days” is calendar days (midnight to midnight), so it can have 86401 seconds after all (86400 'regular' seconds and 1 'leap' second). In that case, Randy argues that the original routine would give the wrong number of seconds; that’s silly. Remove it.

Steve Baird notes a grammar error - “not representable [in]{as} a value of type Time…”

Randy comments on the Image function. It’s primarily for debugging. It was based on RFC-1123, but Pascal has suggested use of ISO 8601. That’s better, and he will update it to use that. Tucker finds a reference to another standard: www.cs.tut.fi/~jkorpela/iso8601.html - at the bottom, discusses CWA 14051-1.

Turning to time zones. It would be better for this to be an offset, which could be passed into yet another overload for Split. It is necessary to pass the time into that function, because the result depends on what time it is for (daylight savings time, for example). The function would have a specification of:

```
function Local_Time_Offset (Date : in Time := Clock) return Time_Offset;
```

Split would take this Time_Offset. Time_Offset would be a number of minutes -1440..+1440.

Make this a separate child (Time_Zones).

Should this go in chapter 9, or annex A? The group feels that it is in chapter 9.

What about the child package name? Even Randy thinks that Operations is lame. No better suggestions surface.

Should we change the definition of Year_Number? Leave the lower bound alone; there is code that uses it to represent No_Time or Unknown_Time, and dates in the distant past aren’t usually interesting. What should the upper bound be? 2399 is suggested. Why not go to 9999? Larger would require more bits in the time representation; let’s not get crazy. 1901..2399.

Approve intent of AI: 10-0-0.

Randy will revise the AI.

AI-352/01 ARINC_653_Processes Profile

Erhard thinks that this seems like a lousy message. If there was some real semantics, perhaps it would be better. But now all it says is “forget the Ada tasking model".

Randy doesn’t like the name of this profile. If you just want sequential programs, why not say Profile (Sequential_Program) to get rid of tasking. Tucker expresses support for this suggestion. But others think this is misleading, as the programs in question are going to use a (non-Ada) form of tasking.

This doesn’t help anyone other than the person writing the Ada runtime. It doesn’t help the user of ARINC. The name gives the impression that the compiler is going to provide some ARINC facilities to the user, but all it does is eliminate functionality.

Tucker thinks that the ARINC model is close enough to Ada’s that it could be mapped to Ada tasks (which should be done in the Ada runtime, not separately.) We don’t want to encourage people to use a model other than Ada tasking — we’d be sending the wrong message to Ada users. Nothing prevents implementers from supporting such a profile if they wish — it doesn't need to be in the standard.

No action: 8-0-2.

AI-353/01 New Restrictions identifier No_Synchronous_Control

There is a spelling error: “Restricitions” in first paragraph.

The wording should be “There are no semantic dependences on package Ada.Synchronous_Task_Control.”

Randy: Remove “applies to …” from the proposal section; this AI should be stand-alone.

Erhard would like more justification for this restriction. Alan explains that for restricted run-time systems, you may want to restrict many things. It’s odd that this is not included in the things that can be restricted. Erhard comments that then we need to add similar restrictions for all of the new packages.

Tucker suggests perhaps a “generic” restriction for this. pragma Restrictions (No_Dependence => “<package_name>); The name must be a language-defined or implementation-defined package. Tucker will write an AI to propose this.

It is pointed out that “No_Asychronous_Control” already exists. We should have this one for consistency.

Approve AI with changes: 10-0-0.

AI-354/01 Group Execution Time Timers

Alan explains the reason for the AI. There is a need for time budgets for groups of tasks.

He notes that there should be a check that a task can belong to at most one group.

What happens when the timer runs out? Does it go negative? Alan thinks it should stay 0; the AI needs to say that.

Steve Baird wonders if you need to be able to find out the identities of the tasks in the group. It was considered but it seemed complex. Tucker suggests a passive iterator generic to provide this ability. An alternative would be to have a function that returns an unconstrained array of task_ids. That seems easier to use.

Alan comments that usually you would need to keep the set of tasks that belong to a group.

Steve Michell wonders what is provided here that couldn’t be provided by code based on existing facilities (including AI-307).

There ought to be a mechanism to identify which group a task belongs to.

Tucker asks what happens when the group timer object goes away. The tasks ought to be removed from the group so they can be added to some other group. When the task terminates, what happens? It should be removed from the group it belongs to (if any).

Randy wonders why this interface is so different than the interface for individual tasks.

If it is called a Group_Timer, the functions should talk about Timers, not budgets.

Why are there two kinds of timers? We already have timers for individual tasks (AI-307). Alan explains that a task might be both monitored by its individual use and its group budget. Alan further explains that the intended use is different; this is a periodic event.

Perhaps it would be better to call this a “budget”, because it is different than timers. Then the different layout of the package doesn’t matter as much.

There is concern about the name Replenish. When you replenish your gas tank, you don’t empty it. Perhaps it should be called Reset. Steve Baird wonders what happens if you replenish with a negative amount. Tucker suggests changing the parameter name to “To”, to make it read better.

Pascal complains that he doesn't like the name "Notify". The type should be called "Handler" (as with interrupts). AI-307 should be changed as well. What should the formal parameter names be? Interrupts use "New_Handler", "Old_Handler". Alan should do something good.

Should this have a restriction for "No_Nested_Budgets"? Tucker would prefer that it simply say that No_Nested_Finalization applies to it. "If the No_Nested_Finalization applies, no nested object has non-trivial finalization." Then, say that the group declared in this package has non-trivial finalization. Pascal will create an AI to do this.

Add a routine to find out whether a task belongs to any group. You should be able to test to avoid an exception.

The implementation of a task requires knowing what group it belongs to. Possibly the budget type should be non-limited. No, that brings up a new can of worms (storage management).

If we have groups, can't we use them for other things? Alan thinks that you'd probably need different groups for different purposes.

When a task terminates, it has to be removed from the group. (Of course, it could have terminated after the group was finalized.) Does this cause a race condition? We probably need "shall be atomic" here. Probably all of the AIs defining new tasking packages need the same sort of wording. Look at C.7.2(16.1) for an example.

Erhard wonders why there aren't more error checks. What if the task id is terminated, or Null_Task_Id, or doesn't point at any task?

Alan will update the AI based on these ideas.

Approve intent of AI: 9-0-1.

AI-355/01 Priority Specific Dispatching including Round Robin

Alan gives us an introduction to all of the new proposals (these are AIs 355, 356, 357, and 358). Each priority level currently uses FIFO. We would like to open that up, and allow round robin and other kinds of dispatching within a priority.

Tucker thinks that is it possible to do Earliest Deadline First scheduling (EDF) with existing facilities. No, that's deadline monotonic scheduling; EDF isn't the same. EDF can find schedules where deadline monotonic scheduling won't. It also is a more efficient use of the hardware.

Turning to the AI, Alan explains that this adds a new kind of policy, Priority_Policy. Tucker wonders if the policy priority is static. Yes, only one of these can be specified for a priority.

The syntax should be "policy_argument_association".

The expected type of the expression is universal-real, and static. The pragma definition needs syntax; nothing about pragmas is predefined. The parameter is supposed to be optional; it needs to be written that way.

Change Nearest_Supported_Quantum to Actual_Quantum.

This shouldn't be a child of Execution_Time. It should be a child under Real_Time called Dispatching; and then Dispatching.Round_Robin.

The function should be is Is_Round_Robin instead of Round_Robin; it's a query.

Alan notes that running out of a quantum is defined similarly to Set_Priority, so that protected operations continue to completion. One can think of this as if protected objects have their assigned priority plus epsilon; thus, they run to completion. There probably needs to be a general statement that pre-emption always puts you at the front of the queue. This should apply to all Priority_Policy schemes.

Alan will update the AI with these suggestions.

Approve intent of AI: 7-0-3.

AI-356/01 Support for Preemption Level Locking Policy

Alan explains for the ideas behind this AI.

Tucker and Pascal comment that this looks like a new mechanism to do something that is already provided by the language (set a PO's priority). Alan says that this is only used for pre-emption, not for scheduling. Ceiling priority is a combination of preemption and scheduling.

The mechanism isn't needed for round-robin, because the quantum doesn't run out while in a PO.

Erhard wonders if you can you get deadline inversion. He tries to explain why; the long deadline task can hold a PO for a long time; because it is preempted by a shorter deadline task. This might block a critical task. Steve wonders if you need deadline inheritance. Alan notes that a rendezvous is a source of possible deadline inversion.

It seems like there is more research here. Tucker asks if deadline inheritance is a problem in practice; Alan says he thinks it isn't important.

Static levels seem to be enough; dynamic seems too complex.

Tucker suggests using a set of priority levels for scheduling, instead of preemption levels. All of the tasks in that set would be scheduled together. Alan thinks that would require more surgery on D.2. But it is very appealing, as it wouldn't require a new model.

After discussing AI-357, we vote on both AIs together. Keep the AIs alive: 6-0-4.

Alan will go back to the drawing board and make another attempt at the AI.

AI-357/01 Support for Deadlines and Earliest Deadline First Scheduling

Tucker notes that there is confusion as to the meaning of deadlines. Alan says that this is an absolute deadline.

Use named notation in the instantiation of Task_Attribute.

The pragma has an absolute time. Steve Baird wonders about the order of evaluation of expressions. How do you set the deadline for a task? It applies to the activation time of the task as well, but you already well into the activation of the task before you elaborate the pragma and evaluate the expression and know the deadline.

Tucker suggests that the pragma be called "Initial_Deadline"; he also would prefer that it is a relative time. Another option would be for it to inherit the deadline from the parent.

Would a relative deadline work? Probably, but would be harder to use than an absolute one.

Tucker asks about activation and rendezvous; you can block at a higher active priority. What scheduling is used for that? If you are pulled into an EDF priority and you don't have a deadline, there is a problem.

Tucker suggests an initial relative deadline for all tasks. The deadline would be relative to the first time it is queued. Then we don't need the pragma at all. So add a parameter to the Priority_Policy to do this.

Tucker wonders what happens if two tasks have the same deadline. Unspecified would be fine, but the AI should say that.

Must revise the model based on active priorities, not base priorities.

After discussing AI-356, we vote on both AIs together. (They're closely related.) Keep the AIs alive: 6-0-4.

Alan will update the AI with these suggestions.

Detailed Review of Regular AIs

AI-331/00 10.1.1(19) doesn't handle multiple nesting?

We look at the problem briefly; it looks like a real problem.

Gary volunteers to take the AI.

AI-332/02 Resolution of qualified expressions

In the question, add (No.) after “IS THIS LEGAL?”

In the AARM Note, add “This is different {from} most other contexts, which allow class-wide types to match specific types.”

Tucker thinks this is the wrong fix. He would rather fix 8.6(27). The problem is that it says the “expected type” and there is none here. He suggests changing the first sentence to: “When the expected type for a construct is required to be a *single* type in a given class, the type [expected] {required} for the construct shall be determinable solely from the context in which the construct appears, excluding the construct itself." That doesn’t help, it still says “expected type”, and there still isn’t one.

[Editor's note: I now agree that 8.6(27) needs to be fixed; I noticed that renames of objects has the same wording (shall resolve to) and thus would have the same problem. That’s less important, but it also ought to be fixed. There probably are other places as well (generic in outs??).]

Assign to the black hole, i.e. Tucker to come up with better wording.