ISO/IEC JTC1/SC22/WG9 N437
ARG Rapporteur's Proposal for Defining Scope of Amendment to ISO/IEC 8652:1995
9 April 2004

Excerpts from document N.412 appear below in **bold font**. My additional comments appear in regular font. I have added a few bulleted subcategories in cases where N.412 only gave broad advice and a few examples.

Because we are arguing about scope, not AIs, I only give AI numbers, not version numbers or dates.

The AIs that are not marked with a symbol have been approved by WG9, or approved by the ARG and will go to the next WG9 meetings.

The AIs that are still in the works are marked with one of the following symbols after their number:

† This AI still needs a bit more work before it can go to WG9, but its technical content is well-defined and believed to be sound (15 AIs).

‡ This AI still needs substantial work before it can go to WG9, and its technical content is still in a state of flux. The reason why it's still alive is that the ARG sees sufficient value in the ideas being proposed, and therefore wants to study them some more before making a final decision. Note that there is no firm consensus on these AIs yet, as many people want to see the AI mature before forming an opinion (7 AIs).

Examples of worthwhile changes are:

- **inclusion of the Ravenscar profile;**

AI95-00249 Ravenscar profile for high-integrity systems
AI95-00265 Partition Elaboration Policy for High-Integrity Systems
AI95-00305 New pragma and additional restriction

AI95-00249 and AI95-00305 together define the language features corresponding to what is known in the vernacular as the "Ravenscar profile". While not strictly part of Ravenscar, AI95-00265 was motivated by practical usage of the Ravenscar profile.

- **inclusion of a solution to the problem of mutually dependent types across packages.**

AI95-00217 Limited With Clauses
AI95-00230 Generalized use of anonymous access types
AI95-00326 Incomplete types

Both AI95-00217 and AI95-00326 are required to solve the problem of mutually dependent types across packages. However, in isolation, they would lead to proliferation of access types and conversions between these types. AI95-00230 addresses this second problem.

The ARG is requested to pay particular attention to the following two categories of improvements:

(A) Improvements that will maintain or improve Ada's advantages, especially in those user domains where safety and criticality are prime concerns;

- **Improvements in the real-time features are an example of (A) and should be considered a high priority.**

AI95-00297† Timing events
AI95-00307† Execution-Time Clocks
AI95-00321 Definition of dispatching policies
AI95-00327† Dynamic ceiling priorities
AI95-00353 New Restrictions identifier No_Synchronous_Control
AI95-00354† Group Execution-Time Budgets
AI95-00355‡ Priority Specific Dispatching including Round Robin
AI95-00356‡ Support for Preemption Level Locking Policy
AI95-00357‡ Support for Deadlines and Earliest Deadline First Scheduling

All these AIs come from the IRTAW, so they are supposed to reflect the needs of the run-time community.

- **Improvements in the high-integrity features are an example of (A) and should be considered a high priority.**

AI95-00266† Task termination procedure
AI95-00347 Title of Annex H

Note that the Ravenscar profile mentioned above is actually a capability that relates to the high-integrity usage of Ada.

- **Features that increase static error detection are an example of (A) and should be considered a priority, but less important than the two listed above.**

AI95-00218 Accidental overloading when overriding
AI95-00231 Access-to-constant parameters and null-excluding access subtypes
AI95-00262 Access to private units in the private part
AI95-00287 Limited Aggregates Allowed
AI95-00310 Ignore abstract nondispatching subprograms during overloading
AI95-00318‡ Returning [limited] objects without copying
AI95-00363† Eliminating access subtype problems

AI95-00218 addresses a problem that can lead to extremely severe errors in systems using OOP, and that is addressed by some other OOP languages (Eiffel, C#).

AI95-00363 eliminates a number of problems with access types which could lead to extremely severe errors.

AI95-00287 and AI95-00318 are intended to make limited types more usable: currently limited types have so many restrictions that they are hardly used at all. If users could use limited type more often, they would benefit from the associated static error detection (in particular to avoid unwanted sharing).

AI95-231 makes it possible to specify more precisely the properties of access types. AI95-262 gives more control on the visibility of entities and makes private units more usable. In both cases, additional static error detection can be obtained by using the new features.

AI95-00310 makes it possible to "undefine" operations, thereby avoiding references to (inherited) operations that don't make sense for an entity.

- **Improvements in the facilities for interfacing to other languages are an example of (A) and should be considered.**

AI95-00216 Unchecked unions -- variant records with no run-time discriminant
AI95-00248 Directory Operations
AI95-00315† Full support for IEC 559:1989
AI95-00351† Time operations
AI95-00370† Environment variables

AI95-00216 is the only AI that actually pertains to interfacing to another language (C). However the ARG felt that there was a need to be able to interface to other *computing environments* as well. The other AIs listed here all address this issue.

- The following AIs add new predefined units which increase the capabilities of the Ada programming environment, and increase the portability of programs:

AI95-00296 Vector and matrix operations
AI95-00302† Container library

(B) Improvements that will remedy shortcomings in Ada.

- **Improvements in the object-oriented features—specifically, adding a Java-like interfaces feature and improved interfacing to other OO languages—are an example of (B) and should be considered.**

AI95-00251 Abstract Interfaces to provide multiple inheritance
AI95-00252 Object.Operation notation
AI95-00345‡ Protected and task interfaces
AI95-00348 Null procedures

Support for Java-like interfaces is provided by AI95-00251, AI95-00345 and AI95-00348. AI95-00252 adds support for a prefix notation which is common in other languages OOP and sometimes more convenient than the traditional Ada notation.

- The following AIs enhance the portability of Ada programs:

AI95-00224 pragma Unsuppress
AI95-00257 Restrictions for implementation-defined entities
AI95-00260 How to control the tag representation in a stream
AI95-00270 Stream item size control
AI95-00286 Assert pragma
AI95-00368† Restrictions for obsolescent features

- The following AIs improve composability of the elements of the language, making it easier to build libraries of components and reuse them:

AI95-00254 Anonymous access to subprogram types
AI95-00317 Partial Parameter Lists for Formal Packages
AI95-00344‡ Allow nested type extensions
AI95-00359‡ Deferring Freezing of a Generic Instantiation

- The following AIs lift somewhat arbitrary restrictions, or add new capabilities that improve the usability and readability of the language:

AI95-00301 Operations on language-defined string types
AI95-00328 Preinstantiations of Complex_IO
AI95-00340 Mod attribute
AI95-00361 Raise with message
AI95-00362† Some predefined packages should be recategorized
AI95-00366† More liberal rule for Pure units

- The following AIs provide mechanisms for improving the efficiency of user programs or of implementations:

AI95-00267 Fast float-to-integer conversions
AI95-00273 Use of PCS should not be normative
AI95-00329 pragma No_Return -- procedures that never return

- The following AIs improve the compatibility between the language as revised by the Amendment and Ada 83 and Ada 95:

AI95-00284 Nonreserved keywords
AI95-00364† Fixed-point multiply/divide

- The following AI improves the support of internationalization and localization:

AI95-00285† Support for 16-bit and 32-bit characters