ISO/IEC JTC1/SC22/WG9 N 666 Instructions to the Ada Rapporteur Group from SC22/WG9

Preparation of Revisions to ISO/IEC 8652:2023

The ARG is instructed to prepare a working draft of a revision to ISO/IEC 8652. The main purpose of the revision is to make needed corrections to the language standard. A secondary purpose is to address identified problems in Ada that are interfering with Ada's usage or adoption, especially in its major applications areas (such as high-reliability, long-lived realtime and/or embedded applications and very large complex systems).

The resulting standard will be published as a Revision, rather than yet another separate document. However, this desire should not trump the desire for language stability (subclauses should continue to use their current numbering, for instance).

The ARG is requested to pay particular attention to the following themes. Although these are described as independent themes, unifying concepts that cross areas are very welcome.

- More convenient and flexible string handling
    - Currently there are three predefined String types in package Standard (String, Wide_String, and Wide_Wide_String) and a collection of packages that support Unbounded, Bounded, and Fixed versions of these, plus some support for converting to and from UTF-8 and UTF-16 representations, etc. The community is looking for something analogous to a "universal" string type which can be used in almost all current contexts where strings are expected (including the various I/O packages), fully supports Unicode, and which is as easy to use as a numeric type.
- Additional light-weight threading capabilities
    - There are various subsystems which are single threaded, but which connect to multithreaded systems via things like promises and call backs. It would be nice if Ada could interact well with such systems, even when the Ada code is using tasking or light-weight parallel constructs.
    - Lock-free programming has been enabled by the Ada 2022 System.Atomic_Operations subsystem. Additional capabilities to support lock-free programming should be considered, such as defining a more

precise memory model together with additional non-blocking synchronization constructs, which might allow higher performance while remaining hardware agnostic.

- Mechanisms for converting arbitrary types to and from various string, stream, or persistent representations.
    - Ada 2022 added a universal 'Image capability supporting type-specific user-defined Put_Image procedures. Clearly an analogous Get_Value procedure would make sense to support 'Value. More generally, the community is looking for simple I/O of complex types via strings, various binary and stream representations (e.g. JSON and XML), as well as mechanisms for supporting persistent file-based representations.
- Better support for heterogeneous and distributed environments
    - Modern computing often involves systems of various architectures, connected either with shared memory or over a high-speed bus. Ada's Distributed System Annex is one approach, but has not been widely implemented nor used. A "building block" approach might be one alternative way to support heterogeneous and distributed systems. This is clearly related to the above item related to representing arbitrary types in some sort of stream representation.
- Simplifying use of Ada's generic units
    - Ada was one of the first languages to support generic units (aka "templates"), but now generics in Ada seem more complex to use than in other languages. Improvements in terms of instantiation on use and inference of generic parameters should be considered as a way to simplify their effective use.
    - Generic parameters might be better integrated with the growing number of type-related aspects, so that one could specify that a given formal type had a particular type-related aspect, such as Variable_Indexing or Aggregate. Enhancements to generic parameter capabilities might be helpful in addressing the multiple different string types currently provided by Ada.
- Further unifying tagged and untagged types
    - Recent versions of the Ada Standard have reduced the number of places where a tagged type is required to get certain functionality. This might be continued, both to improve uniformity, and in some cases reduce the overhead. Examples might include finalization, storage management,

object-dot-operation notation, user-defined indexing, etc. (Some of these enhancements have already appeared as Ada 202Y Ada Issues.)

- Safe, ownership-based automated resource management
    - The Rust language and recent versions of the SPARK subset of Ada have provided an ability to automate resource management without requiring the use of a garbage collector, reference counting, or potentially expensive run-time finalization, by building on the notion of object or pointer "ownership". This could be a good fit for resource-constrained applications which cannot afford the uncertainty and/or overhead of run-time garbage collection or finalization.

In selecting features for inclusion in the revision, the ARG should consider the following factors:

- Implementability (vendors' concerns).
    - Can the proposed feature be implemented at reasonable cost?
- Need (users' concerns).
    - Does the proposed feature fulfill an actual user need?
- Language stability (users' concerns).
    - Would the proposed feature appear disturbing to current users?
- Competition and popularity.
    - Does the proposed feature help improve the perception of Ada, and make it more competitive with other languages?
- Interoperability.
    - Does the proposed feature ease problems of interfacing with other languages and systems?
- Language consistency:
    - Is the provision of the feature syntactically and semantically consistent with the language's current structure and design philosophy?

In order to produce a technically superior result, it is permitted to minimally compromise backwards compatibility when the impact on users is judged to be acceptable. The use of secondary standards should be minimized; secondary standards should be proposed only when they would include material so important as to require standardization, but too voluminous to be included in the Ada language standard.