ADA 95 AND SAFETY-CRITICAL SOFTWARE

C. Michael Holloway

NASA Langley Reseach Center
Assessment Technology Branch
Mail Stop 130
Hampton, VA 23681-0001
Email: C.M.Holloway@LaRC.NASA.GOV

ABSTRACT

The revised Ada standard (ISO/IEC-8652:1995, commonly referred to as Ada 95) was released by the International Standards Organization (ISO) in February 1995. One of the unique features of this standard is that it is divided into a Core Language, which must be fully implemented, and several Specialized Needs Annexes, which provide standard definitions for additional features for particular application areas.

Of particular interest to developers of safety-critical software is Annex H: Safety and Security. This Annex specifies detailed documentation requirements and facilities to support enhanced understanding of program execution paths and for reviewing object code. It also provides facilities for restricting the use of certain language constructs. This paper will discuss Annex H and will also present the charter of the recently formed Annex H Rapporteur Group.

INTRODUCTION

The International Standards Organization (ISO) published the revised Ada standard [1] and an accompanying Rationale [2] in February 1995. The new standard replaces the 1987 ISO standard [3], which was identical to the 1983 American National Standards Institute (ANSI) standard [4]. To distinguish the language defined by the new standard from that defined by the older standards, the names Ada 95 and Ada 83 are frequently used; this naming convention is adopted here[1]. The primary purpose of this paper is to discuss the provisions made in the Ada 95 standard to support the development of safety-critical software.

To accomplish this purpose, the paper is organized in the following manner. First, the history of the revision process is reviewed. Second, the specific safety-related requirements developed for the language revision are enumerated. Third, the details of the Ada 95 Safety and Security Annex (Annex H) are discussed. Fourth, the charter of the recently established Annex H Rapporteur Group (HRG) is presented. Fifth, concluding remarks are made.

THE REVISION PROCESS

Before discussing the particulars of the Ada 95 provisions for developing safety-critical software, a brief discussion of the history of the revision process is appropriate. The discussion below is based on [5]; the interested reader is encouraged to consult this document for additional details not included here.

The Ada programming language was originally developed to meet the specific requirements of the United States Department of Defense (DOD) for a single language for real-time embedded systems [6]. However, after the ANSI standard was released in 1983, interest in the language grew beyond a single application domain and a single agency. International use of Ada was such that the ISO also adopted the ANSI standard.

In 1988, as part of the routine 5-10 year reexamination of existing standards, the decision was made to revise the Ada language, with the intent of creating a new, joint ANSI/ISO standard. ANSI sponsored the revised standard, with the DOD managing the development through the Ada Joint Program Office (AJPO). The AJPO established the Ada 9X Project Office, and named Christine Anderson as the project manager in October 1988.

[1] Whether this particular convention is an appropriate one has been the subject of several amusing discussions in the comp.lang.ada Internet newsgroup.

The Ada 9X Project Plan was released in January, 1989 [7]. The plan stated the goal of the revision process clearly and succinctly:

> *The overall goal of the Ada 9X Project is to revise ANSI/MIL-STD-1815A to reflect current essential requirements with minimum negative impact and maximum positive impact to the Ada community.*

According to the plan, the goal was to be achieved in three main phases: determination of the essential requirements for the revised language; the mapping of these requirements onto language changes followed by the adoption of the revised language by all major standards organizations; and the transition in use from Ada 83 to Ada 9X.

The determination of essential requirements began with a Workshop in June 1989 [8], included additional workshops and solicitation of revision requests from users world-wide (over 750 were received), and culminated in the publishing of a requirements document in December 1990 [9] and an associated rationale in May 1991 [10].

The requirements document established 41 Requirements and 22 Study Topics, and directed the mapping/revision team to attempt to satisfy all of the Requirements and as many of the Study Topics as possible. In recognition of the specialized needs of particular application areas, the document recommended that the revised language definition be divided into a Core language, which all validated compilers must implement, and a small number of specialized Annexes, which may or may not be implemented.

The mapping/revision team followed the recommendations in the requirements document and the final, approved language definition includes a Core language (Sections 1-13, Annexes A, B, and J) and the following Specialized Needs Annexes:

- Annex C: Systems Programming
- Annex D: Real-Time Systems
- Annex E: Distributed Systems
- Annex F: Information Systems
- Annex G: Numerics
- Annex H: Safety and Security

These specialized needs annexes provide standard definitions for additional features for particular application areas. Many of the features described in these annexes are provided by various Ada 83 compiler vendors, but in non-standard ways. The Ada 95 standard provides a means to reduce the non-uniformity among compilers for important domains.

SAFETY REQUIREMENTS

Although many parts of the Ada 95 definition have an impact on the development of safety-critical software, our focus in this paper is on the particular features defined in Annex H. These features were designed to address one Study Topic and three Requirements identified in the Ada 9X Requirements. These are enumerated below, under the headings used in the Requirements document.

Predictability of Execution

Although in many application domains, determining the precise behavior of the execution of a program may be unnecessary so long as the program produces the appropriate output, developers of safety-critical applications[1] often must be able to determine precise behavior by examination of the program text. Execution must be predictable. To address this need, one Study Topic and one Requirement were identified.

Study Topic S9.1-A(1) Determining Implementation Choices — *Wherever Ada 9X explicitly allows implementation-defined choices that affect program behavior, implementations shall be required either to document the choice that has been made (or that situations that control what choice is made) or Ada 9X shall provide a mechanism for controlling the choice.*

Requirement R9.1-A(2) Ensuring Canonical Application of Operations— *Ada 9X shall provide a mechanism, applicable to a region of program text, for restricting any freedom otherwise allowed to reorder, replace, or remove actions involving predefined operations.*

Certifiability

The complexity of translating a high-level programming language into object code almost guarantees that any translator will be less than 100 per cent

[1] Nearly all that is said here about safety-critical applications also applies to secure applications, which is why both the requirements document and the language standard group safety and security together. Because the focus of this paper is on safety, mention of security-related aspects is omitted.

accurate. As a result, developers of safety-critical applications tend to certify software correctness based on generated object code, and not on high-level language source code. In such cases, the source code is treated as documentation for the object code, and clear correspondence between the two is required. The following requirement addressed this need.

Requirement R9.2-A(1) Generating Easily Checked Code— *Ada 9X shall provide a mechanism for advising a compiler that code should be generated in a style that allows it to be checked against the source text with reasonable effort.*

Enforcement of Safety-Critical Programming Practices

Some features of a programming language that are perfectly acceptable for many applications may be inappropriate for safety-critical applications. Nondeterminism is one example, and dynamic storage allocation is another. As a result, projects developing safety-critical software typically restrict the language features that may be used. The following requirement addressed this need.

Requirement R9.3-A(1)— *Ada 9X shall allow a mode in which a compiler enforces adherence to coding practices beyond those imposed by the rules of the language.*

ANNEX H

In order to address the above mentioned requirements, Annex H provides five specific facilities: pragma Normalize_Scalars, imposition of additional documentation requirements, pragma Reviewable, pragma Inspection_Point, and a set of restrictions to be used with pragma Restrictions[1]. Each of these facilities is discussed below.

Pragma Normalize_Scalars

The syntax of this pragma is as follows:

pragma Normalize_Scalars;

The pragma requires that each scalar object that is not explicitly initialized in the source text is initialized by the compiler to some documented default value. Whenever possible, the default value should be an invalid (that is, out-of-range) value.

[1] Pragma Restrictions is part of the Core language; it is defined in section 13.12.

The purpose of the pragma is two-fold: (1) to provide predictable behavior in the presence of uninitialized scalars, and (2) to assist programmers in locating and correcting instances of uninitialized scalars. The pragma partially addresses the issues raised in Study Topic S9.1-A(1).

Documentation Requirements

To address those aspects of Study Topic S9.1-A(1) not addressed by pragma Normalize_Scalars, the Annex requires that implementations *"document the range of effects for each situation that the language rules identify as either a bounded error[2] or as having an unspecified effect."* Three examples of situations to which this requirement applies are discussed below.

Parameter Passing Mechanism — Ada language rules permit some parameters to be passed either by reference or by copy. In fact, an implementation may choose a different mechanism for different calls of the same subprogram. To comply with Annex H, a compiler must indicate the parameter passing mechanism chosen for all calls.

Storage Management— an implementation must document the storage management procedures used so as to permit review of the object code to ensure that any applicable storage restrictions are not violated.

Evaluation of Numeric Expressions — evaluating numeric expressions can produce widely varying results when the evaluation involves using extended ranges or extra precision. For this reason to comply with Annex H, an implementation must so document the evaluation approach used, that for any given expression, the range and precision with which it is computed is clear.

Pragma Reviewable

The syntax of this pragma is as follows:

pragma Reviewable;

The pragma partially addresses Requirement R9.2-A(1). It requires the compiler to provide adequate information for analysis and review of generated object code; this information is to be provided in both a machine and a human readable form. Some of the

[2] A bounded error is an error that the language rules do not require to be detected by implementations, but for which the rules do require that the range of possible effects to be bounded.

particular types of information that must be provided are discussed below.

Elaboration Order for Library Units — the precise order in which units are elaborated must be clear.

Object Lifetime Analysis — Sufficient information must be supplied about each object to enable a user to determine which objects are assigned to which registers and for how long each assignment lasts.

Initialization Analysis — Each reference to a scalar object must identify whether the object is "known to be initialized" or "possibly uninitialized". This requirement is not superceded by the use of pragma Normalize_Scalars.

Machine Instructions Used — A list must be provided of all the machine instructions used.

Source and Object Code Relationship — For each declaration and statement in a source program, the corresponding generated object code sequences must be identified. If a particular source code statement results in no object code, this must be explicitly identified.

Exception Analysis — All compiler generated run-time checks in the object code must be identified. Also, each construct must be identified for which the implementation detects the possibility of erroneous execution.

Pragma Inspection_Point

To further address Requirement R9.2-A(1) and to also partially address Requirement R9.1-A(2), the pragma Inspection_Point is provided. The syntax of the pragma is as follows:

pragma Inspection_Point [(name {, name})];

where each name must denote the declaration of an object.

This pragma provides a means for specifying points in the program text at which the values of particular objects must be available. In particular, for each inspection point, at the corresponding point(s) in the object code, a means must be provided for determining the values of the specified objects (or, if no objects were specified, of all live objects).

One interesting aspect of inspection points is that they provide some of the capabilities of an assertion facility. As an example, suppose at some point in a program we know that the value of object X should be strictly greater than the value of object Y, then we can write

pragma Inspection_Point (X, Y);

at the appropriate point in the code. If we execute the program using a suitable debugger, it can be suspended at the point(s) in the object code corresponding to this inspection point. We can then examine the values of X and Y and determine if X is strictly greater than Y[1].

Inspection points also provide a mechanism that can be used by special tools to analyze and verify particular properties of the object code. In fact, given a suitable mathematical specification, an adequate tool, and appropriate use of inspection points, a partial or full formal mathematical verification of object code would be possible.

Allowed Restrictions

The final facility defined in Annex H addresses Requirement R9.3-A(1) by providing a means by which use of particular language features may be prohibited. The Core Language includes a pragma Restrictions, with the following syntax:

pragma Restrictions (restriction {, restriction});

where restriction is either an identifier or of the form:

identifier => expression

The Annex defines a number of possible restrictions that may be given, including some that are also defined (identically) in the Annex for Real-Time Systems (Annex D). The following are a few of the possible restrictions along with their meanings:

- No_Exceptions: Prohibits using raise statements and exception handlers and ensures that no language-defined run-time checks are generated.
- No_Floating_Point: Prohibits using predefined floating point types and operations and declarations of new floating point types.
- No_Allocators: Prohibits using allocators.
- No_Access_Subprograms: Prohibits the declaration of access-to-subprogram types.

[1] Assertions were provided in an early draft of the Annex, but they were ultimately removed because of the many subtleties involved in their implementation.

- No_Dispatch: Prohibits, for any tagged subtype T, occurrences of T'Class.
- No_IO: Prohibits semantic dependence on any of the library IO units.
- No_Delay: Prohibits delay statements and semantic dependence on package Calendar.

The Annex also defines the following restrictions (see the Annex for their definitions):

- No_Task_Hierarchy
- No_Abort_Statement
- No_Implicit_Heap_Allocation
- Max_Task_Entries
- Max_Asynchronous_Select_Nesting
- Max_Tasks
- No_Protected_Types
- No_Local_Allocators
- No_Unchecked_Deallocation
- Immediate_Reclamation
- No_Fixed_Point
- No_Unchecked_Conversion
- No_Unchecked_Access
- No_Recursion
- No_Reentrancy

ANNEX H RAPPORTEUR GROUP

In recognition of the importance of safety-critical software and of Annex H, the ISO working group responsible for the Ada standard (WG9) established the Annex H Rapporteur Group (HRG) in April, 1995. The charter of the group is as follows.

HRG Charter Approved by WG9, 28 April 1995—
The HRG will synthesize the essential requirements of typical sector-specific standards for high integrity applications which have a bearing on Ada and its supporting tools. Guidance, including interpretation and amplification of Annex H will be developed for users, implementers, evaluators and certifiers. The guidance produced will be in a form suitable for reference in procurement.

Sector-specific standards to be considered are such as:

- DO-178B (Civil avionics)
- IEC 65A/CENELEC (Generic/rail)
- IEC 880 (Nuclear)
- Interim DEFSTAN 00-55 (UK Defence)
- ITSEC (EU Security)

The HRG will undertake the following activities:

Annex H Issues

The HRG will produce and maintain an interpretations document.

The HRG will investigate pragma enhancement, such as additional parameters for restriction pragmas and additional pragmas.

The HRG will provide implementation advice, covering areas such as compilation and validation.

Taxonomy of Techniques

The HRG will produce a taxonomy of techniques for the construction and analysis of high integrity software, such as:

- The use of annotations in program construction
- Error detection by static analysis
- Design confirmation by static analysis
- Static timing analysis

Language Issues

The HRG will investigate the interaction of language issues with high integrity requirements, such as:

- Deterministic execution with compiler optimization and other property-based subsets
- Concurrency
- Issues of migration from Ada 83

Bindings and Interfaces

The HRG will support the interoperation of high integrity software and tools with other systems, such as:

- ASIS (Ada Semantic Information Specification)
- Ada compilers and run-time environments
- CORBA (Common Object Request Broker Architecture)

Anyone interested in more information about the HRG should contact Brian Wichmann of the National Physical Laboratory, United Kingdom (E-mail: baw@ditc.npl.co.uk). The next meeting of the group is scheduled for September 14-15, 1995.

CONCLUDING REMARKS

This paper has presented an overview of some of the facilities provided in Ada 95 to help support the development of safety-critical software. In particular, the paper has concentrated on those facilities defined in Annex H of the language standard. The discussion has presented an overview only; those readers interested in additional details should consult the referenced documents directly. Readers with access to the World-Wide Web might want to consider browsing the following URL's:

- **http://lglwww.epfl.ch/Ada/LRM/9X/rm9x/rm9x-H.html** -- text of Annex H.
- **http://lglwww.epfl.ch/Ada/LRM/9X/Rationale/rat95html/rat95-p3-h.html** -- text of the Rationale for Annex H.
- **http://www.npl.co.uk/npl/ditc/seg/hrg/infopack.html** -- information on the HRG
- **http://atb-www.larc.nasa.gov/fm.html** -- this URL contains information on the formal methods program at NASA Langley Research Center, which includes work related to Ada and to safety-critical software.

REFERENCES

[1] ANSI/ISO/IEC-8652:1995. *Ada 95 Reference Manual*, January 1995.

[2] Intermetrics, Inc., Cambridge, Massachusetts. *Ada 95 Rationale*, January 1995.

[3] ISO/8652:1987. *Reference Manual for the Ada Programming Language*, 1987.

[4] American National Standards Institute, ANSI/MIL-Std-1815a. *Reference Manual for the Ada Programming Language*, 1983.

[5] J. Barnes. *Introducting Ada 9X*. Ada 9X Project Office, Office of the Under Secretary of Defense for Acquisition, Washington, DC, February 1993.

[6] Defense Advanced Research Projects Agency, United States Department of Defense, Arlington, Virginia. *Department of Defense Requirements for High Order Computer Programming Languages: STEELMAN*, 1978.

[7] Ada 9X Project Office, Office of the Under Secretary of Defense for Acquisition, Washington, DC. *Ada 9X Project Plan*, January 1989.

[8] Ada 9X Project Office, Office of the Under Secretary of Defense for Acquisition, Washington, DC. *Ada 9X Requirements Workshop*, June 1989.

[9] Ada 9X Project Office, Office of the Under Secretary of Defense for Acquisition, Washington, DC. *Ada 9X Requirements*, December 1990.

[10] Ada 9X Project Office, Office of the Under Secretary of Defense for Acquisition, Washington, DC. *Ada 9X Requirements Rationale*, May 1991.