GSFC Ada Programming Guidelines

Daniel M. Roy, Robert W. Nelson

1 INTRODUCTION

A significant Ada effort has been under way at Goddard for the last two years. To ease the center's transition toward Ada (notably for future space station projects), a cooperative effort of half a dozen companies and NASA personnel was started in 1985 to produce programming standards and guidelines for the Ada language.

2 APPROACH

Two parallel tracks were pursued:

1. Coding style and Ada statement format.
2. Portability, efficiency and whole life cycle issues.

Two documents have been produced so far, one for each track followed. This paper more specifically deals with the second one. Both documents are similar in structure (closely modeled on the Ada LRM) and were greatly influenced by Nissen and Wallis guidelines ([NW]). Other documents also had some influence:

- The rationale for Ada [Rationale].
- The IEEE Ada PDL recommended practices document [IEEE-990].
- Intermetrics BYRON user's guide [Intermetrics].
- Ada in practice (Ausnit, Cohen, Goodenough, and Eanes) [Softech].
- Using Selected Features of Ada [NTIS].
- Intellimac's Ada style [Intellimac].
- Regulation for the management of computer resources in defense systems (MIL-STD-2167) [2167].

Both drafts are currently being merged into an Ada Style document for use by all projects at the NASA Goddard Space Flight Center.

3 STRUCTURE OF THE DOCUMENT

It was decided early on to model our guide on the Ada Language Reference Manual (LRM) for the following reason:

1. The LRM gives us a frame of reference that is a standard.

2. By following the LRM, we can reasonably expect to be thorough.

3. We intend to illustrate the LRM jargon with good Ada code examples.

Therefore, the document follows the numbering of the LRM as closely as possible, including the appendices. However, in spite of this convention, our Ada Programming Guidelines are sufficiently self contained that they can be read without the LRM.

Chapters 1 to 14 of our document closely follow the corresponding LRM sections.

Appendix A of the document (Language Attributes in the LRM) describes the recommended documentation keywords both for design (user oriented) and code (programmer oriented).

Appendix B of the document (Predefined Pragmas in the LRM) illustrates the usage of pragmas.

Appendix C of the document (Predefined Language Environment in the LRM) gives the Ada source code of a decision deferral package (package TBD).

Appendix D of the document (Glossary in the LRM) is a glossary of terms used in the guide and not defined in the LRM.

Appendix E of the document (Syntax Summary in the LRM) is a place holder for the definition of "Ada_LINT", an Ada style and programming practice analyser. After a consensus has been reached about the specification of the tool and its command language, this appendix will include:

1. The APSE tool command language syntax and semantics definition.

2. The directives embedded in Ada documentation, style specification files, etc.

Appendix F (Implementation Dependent Characteristics in the LRM) identifies the links, waivers or modifications to the company standards made necessary by these guidelines.

Appendix G is a place holder for the definition of a "pretty printer" utility. After a consensus has been reached about the specification of the tool and its command language, this appendix will include:

1. The APSE tool command language syntax and semantics definition.

2. The directives embedded in Ada documentation, format specification files, etc.

Appendix H is an annotated bibliography.

The illustrated, recommended practices and guidelines suggest rules and provide examples of good Ada design and coding formats to promote readability, maintainability and, therefore, portability and reusability of Ada code.

An effort was made to alleviate the bureaucratic burden (that so often mars software standards) by concentrating on the programmer's "need to understand" and relying on automated tools for the mechanical (and subjective) aspects of programming such as indentation, alignment of tokens, etc. Most such rules are to be localized in an Appendix (Pretty_printer Definition).

Automated support from simple code templates and comment constructs to the definition of APSE tools are also considered.

4 EXCERPTS FROM THE GUIDELINES

Figure D.1.4-1 introduces the recommended comment constructs that allows simple tools to extract PDL or documentation from the Ada design or code.

The document strives to complement the LRM by illustrating its jargon with examples whenever possible. Unless the rule is particularly obvious, a rationale is given (possibly in the form of a bibliography reference), and an example is proposed. The rules are classified as either suggestions or strong recommendations. The latter are underlined for emphasis.

Figure D.1.4-2 to D.1.4-5 show the typical format of the rules given.

The document also draws on the IEEE 990 document (Ada as a Design Language) to show the smooth progression from Ada design to Ada code where practical. Figures D.1.4-6 and D.1.4-7 show two examples adapted from the IEEE document.

Finally, because efficiency issues pervade the LRM, the guide addresses the tradeoffs between readability, portability and

efficiency where appropriate.

5 CONCLUSION

The great richness of the Ada language and the need of programmers for good style examples, make Ada programming guidelines an important tool to smooth the Ada transition.

Because of the natural divergence of technical opinions, the great diversity of our government and private organizations and the novelty of the Ada technology, the creation of an Ada programming guidelines document is a difficult and time consuming task. It is also a vital one.

Steps must now be taken to ensure that the guide is refined in an organized but timely manner to reflect the growing level of expertise of the Ada community.

Daniel Roy is a senior member of the technical staff at Century Computing Inc. where he has been working since 1983. He received the Diplome d'Ingenieur Electronicien (MSEE) from ENSEA in 1973 and the Diplome d'Etudes Approfondies en Informatique (MSCS) from the University of Paris VI in 1975.

Robert W. Nelson is a member of the technical staff in the Software Engineering Section at NASA's Goddard Space Flight Center. He received a B.S in Mathematics from Drexel Institute of Technology and an M.S. in Numerical Science from Johns Hopkins University.

Authors current address:
Century Computing, Inc., 1100 West street, Laurel, Md., 20707.
Tel: (301) 953 3330.

Goddard Space Flight Center, Code 522, Greenbelt, Md. 20771.
Tel: (301) 344 4751.

2.7 COMMENTS

Comments should convey information not directly expressible in Ada. The conventions given below are used throughout this document.

(a) Use "--|" to indicate documentation [Intermetrics].

See Appendix A for the recommended documentation template.

(b) Use "--*" to indicate PDL construct [Intermetrics].

Using Ada as a PDL has numerous advantages. See [IEEE-990].

In the example of a function stub below, the three lines of the function specification are both documentation and PDL.

```
subtype INQUIRED_VAR_TYPE is TBD.SOME_TYPE;
function INQUIRE_INT (          --| Emulate DCL verb for integers --*
  PROMPT : STRING               --| --*
  ) return INQUIRED_VAR_TYPE is       --| --*

  type TRY_RANGE is range 1 .. TBD.MAX;       -- Nr try
  INQUIRED_VAR : INQUIRED_VAR_TYPE := 0;      -- Value returned
--
begin --* INQUIRE_INT
  --* Displays "prompt (min..max): "
 ERROR_LOOP:  --* Until good data or nr errors > max
  for TRY in TRY_RANGE loop   --*
    --* Get unconstrained value
    --* Validate and translate unconstrained value
    return INQUIRED_VAR ;     --*
  end loop ERROR_LOOP;        --*
end INQUIRE_INT ;     --*
```

See Appendix C for the definition of the decision deferral package (Package TBD).

Figure D.1.4-1: Rule for comments.

3.2.2 Number declarations

(a) Do not use numeric literals except in a constant declaration or when a number is obviously more appropriate.

This yields more readable and more maintainable code since a change in value will be localized to the constant declaration.

```
-- Circle object characteristics
RADIUS : constant := 10.0;              -- meters (constant object)
PI : constant := 3.14159;               -- (This is a named number)
CIRCLE_AREA := PI * (RADIUS ** 2);      -- (2 better than "TWO")
```

As a rule, using a constant object is better than using a named number which itself is better than using a numeric literal [NW].

Figure D.1.4-2: Illustrating the LRM jargon.

4.4 EXPRESSIONS

(a) Use parentheses to enhance the readability of expressions [NW].

```
X := (A + B) * (C / ((D ** 2) + E));
```

(b) Use static universal expression for constant declaration [NW].

Universal expressions maximize accuracy and portability. Static expressions eliminate run time overhead.

```
SMALL_STUFF : constant := 12  -- Better than "constant INTEGER :="
KILO : constant := 1_000;
MEGA : constant := KILO * KILO;
```

Note that the declaration of object "MEGA" would be less portable had KILO been declared as INTEGER since INTEGER'LAST could be less than one million on some target systems.

Also note that the following declarations are more readable than they would be using the constants MEGA and KILO above.

```
type MASS_TYPE is FLOAT range 1.0 .. 1.0E12;  -- Grams
GRAMS : constant MASS_TYPE := 1.0;
KILOGRAMS : constant MASS_TYPE := 1_000.0 * GRAMS;
TONS : constant MASS_TYPE := 1_000.0 * KILOGRAMS;
```

Figure D.1.4-3: Discussing the rules.

CHAPTER 9

TASKS

(a) Use a task for:

- o modeling concurrent objects (such as airplanes in an airport simulation).
- o asynchronous IO (other tasks may run while the IO task is blocked).
- o buffering or providing an intermediary link between asynchronous activities (buffer, active link between two passive tasks).
- o hardware dependent, application independent functions (device drivers, interrupt handlers).
- o hardware independent, application dependent functions (monitors, periodic activity, activity that must wait a specified time for an event, vigilant activity, and activity requiring a distinct priority).
- o programs that run on a distinct processor.

It is imperative that the methodology selected to develop multitasking systems minimize the number of tasks and provide guidance in the usage of the numerous tasking features of Ada. See [Cherry-84] for details.

Figure D.1.4-4: Rules and bibliography.

(b) Encapsulate priorities in a package [NW].

The LRM does not specifiy the number of priority levels.

```
with SYSTEM;use SYSTEM;        -- Makes sense here to shorten declarations
package PRIORITY_LEVELS is     --| Implementation dependent
--| Raise:
--|    The following declarations can raise CONSTRAINT_ERROR on
--|    some implementations since the number of priority levels
--|    is not defined in the LRM.
--| Purpose:
--|   Encapsulate implementation dependent priority definitions.
--| Portability:
--|   Some declarations may have to be modified for systems featuring
--|   less than 16 levels.  For instance *_HIGH and *_MED priorities
--|   may have to become equal to *_LOW in an 8 levels system.
--| Notes:
--|   Change Log:
--|   Daniel Roy      1-mar-86        Baseline

  LOWEST : constant PRIORITY := PRIORITY'FIRST;
  HIGHEST : constant PRIORITY := PRIORITY'LAST;
  NR_PRIORITY_LEVELS : constant POSITIVE := HIGHEST - LOWEST + 1;
  AVERAGE : constant PRIORITY := NR_PRIORITY_LEVELS / 2;

  IDLE : constant PRIORITY := LOWEST;
  BACKGROUND_LOW : constant PRIORITY := AVERAGE - 6;
  BACKGROUND_MED : constant PRIORITY := AVERAGE - 5;
  BACKGROUND_HIGH : constant PRIORITY := AVERAGE - 4;
  USER_LOW : constant PRIORITY := AVERAGE - 3;
  USER_MED : constant PRIORITY := AVERAGE - 2;
  USER_HIGH : constant PRIORITY := AVERAGE - 1;
  FOREGROUND_LOW : constant PRIORITY := AVERAGE + 1;
  FOREGROUND_MED : constant PRIORITY := AVERAGE + 2;
  FOREGROUND_HIGH : constant PRIORITY := AVERAGE + 3;
  SYSTEM_LOW : constant PRIORITY := AVERAGE + 4;
  SYSTEM_MED : constant PRIORITY := AVERAGE + 5;
  SYSTEM_HIGH : constant PRIORITY := AVERAGE + 6;
end PRIORITY_LEVELS;  --|

-- Using priorities
with PRIORITY_LEVELS;
task NASCOM_SERVER is --| Distribute NASCOM blocks
  pragma PRIORITY (PRIORITY_LEVELS. SYSTEM_LOW);
.....
end NASCOM_SERVER;
```

Figure D.1.4-5: Adding to Nissen and Wallis.

10.2.1 Example of subunits

The following example is adapted from [IEEE-990] and shows how to defer decisions at design time, using Ada as a PDL.

```
with TRACKER_DATA_TYPES; use TRACKER_DATA_TYPES;
procedure TARGET_TRACKER is   --| Radar echo processing

  ECHO : ECHO_TYPE;
  SMOOTHED_RANGE : SMOOTHED_RANGE_TYPE;
  SMOOTHED_ANGLES : SMOOTHED_ANGLES_TYPE;

  package FILTERING_ALGORITHMS is      --| Could be later extracted from
                                       --| here and "with'ed"
    function RANGE_SMOOTHING (         --|
      RAW_ECHO : ECHO_TYPE             --|
      ) return SMOOTHED_RANGE_TYPE;    --|

    function ANGLES_SMOOTHING (        --| May be a generic SMOOTHING
      RAW_ECHO : ECHO_TYPE             --| function could be written.
      ) return SMOOTHED_ANGLES_TYPE;   --|

  end FILTERING_ALGORITHMS;            --|

  -- The following postpone implementation decisions
  -- Simple stubs could be written
  function IS_ECHO_VALID (             --|
    RAW_ECHO : ECHO_TYPE               --|
    ) return BOOLEAN is separate;      --|
  package FILTERING_ALGORITHMS is separate;

begin --* TARGET_TRACKER
  .....
  if IS_ECHO_VALID (ECHO) then         --*
    SMOOTHED_RANGE := FILTERING_ALGORITHMS. RANGE_SMOOTHING (ECHO); --*
    SMOOTHED_ANGLES := FILTERING_ALGORITHMS. ANGLES_SMOOTHING (ECHO); --*
  else        --* decoy ?
    --* log decoy candidate coordinates
    null;
  end if;     --* IS_ECHO_VALID
  .....
end TARGET_TRACKER;   --|
```

Figure D.1.4-6: Using subunits and the TBD package.

Note that all types from the TRACKER_DATA_TYPES package may have been fully described (using Ada as a data definition language and TRACKER_DATA_TYPES as a data dictionary). Another solution is to use the TBD package:

```
with TBD;
package TRACKER_DATA_TYPES is --| data dictionary
--| Notes:
--|   Preliminary design

  subtype ECHO_TYPE is TBD.RECORD_TYPE;
  subtype SMOOTHED_RANGE_TYPE is TBD.REAL_TYPE;
  subtype SMOOTHED_ANGLES_TYPE is TBD.ARRAY_TYPE;
  .....
end TRACKER_DATA_TYPES;        --|
```

Figure D.1.4-6 (cont.): Using subunits and the TBD package.

(b) Use generics as a decision deferral technique during design. [IEEE-990]

```
generic                          --| Decision deferral
  type LIST_TYPE is private;     --| Don't want to bother with details now
function SORT (                  --|
  LIST : LIST_TYPE               --|
  ) return LIST_TYPE;            --|
--| Notes:
--|   Preliminary design

function SORT (                  --| --*
  LIST : LIST_TYPE               --| --*
  ) return LIST_TYPE is          --| --*
--| Notes:
--|   Preliminary design stub
  SORTED_LIST : LIST_TYPE;
begin --* SORT
  SORTED_LIST := LIST;
  return SORTED_LIST;            --*
end SORT;                        --| --*
```

The above generic unit can be further refined at detailed design time using the same kind of technique:

```
-- Adapted from [IEEE-990]
generic                          --| Decision deferral
  type ELEM_TYPE is private;     --| Members of the list
  type INDEX_TYPE is (<>);       --| Can be INTEGER or ENUMERATION type
  type LIST_TYPE is array (      --| We know more about type now
      INDEX_TYPE range <>        --|   but we still defer decisions
      ) of ELEM_TYPE;            --|   about index and element types
  with function "<" (            --| We now know we'll need to overload "<"
    LEFT : ELEM_TYPE;            --|   for our type.
    RIGHT : ELEM_TYPE            --|
    ) return BOOLEAN;            --|
function SORT (                  --|
  LIST : LIST_TYPE               --|
  ) return LIST_TYPE;            --|
--| Notes:
--|   Detailed design
```

Figure D.1.4-7: Using generics to defer decision.