Object Magazine Preprint, May 1997

Ada 95: The Language Speaks for Itself

ROBERT C. LEIF, TOM MORAN, AND RANDY BRUKARDT

Why Ada?

Ada 95, because of its superb facilities for object oriented programming (OOP), presents an interesting challenge to the software industry, particularly here in the United States. Are we going to embrace total Quality Management or be governed by marketing hype? Cool is not an engineering term. Since we wish to emphasize the benefits of Ada, we will describe our technology. We will leave it to the JAVA converts to describe the sins of their previous language.

As most engineers and artists are painfully aware, it is extremely improbable to get something right the first time. Surprisingly, the first ISO (International Organization for Standardization) standardized version of Ada was sufficiently well designed that most of the changes for the second version were extensions rather than revisions. Most of the competing object-oriented languages are immature, since they have not even completed their first round of ISO standardization.

It has been argued in this magazine[1] that "Although the language (Ada) has been with us more than a decade, it has sparked little interest in the private sector". The new Boeing 777 and commercial satellite systems are significant counter examples. Ada 95 is still quite new. However, the best evidence that it often takes considerable time for a new language to be adopted in the commercial sector is Smalltalk, which predates Ada. Smalltalk has only caught on in the last few years.

The authors of this article will prove our point by showing below an example of Ada code for financial transactions with Microsoft Windows™. Dear reader, even if you have never seen Ada before, you will notice that the source code is eminently readable. This is a required characteristic of a document that will be of use to a maintenance programmer, who needs to determine what the program is doing. Source code is an artifact that is used by humans; computers prefer binary. In fact in the case of Ada, the term "Source Code" is a misnomer, which should be replaced by the term "Source Text". Experience in cryptography is not a requirement for reading and understanding Ada.

Because Ada syntax is derived from Pascal, programmers trained in the Pascal family of languages have a significantly easier time learning Ada than members of the C family of languages. Ada's Pascal heritage includes the use of the equal sign (=) for equality and (:=) for assignment (set equal to). Unlike C and its descendants including C++ and JAVA, where the accidental repetition of a single character can result in a day or more of debugging, in Ada, a repeated character almost always results in the compiler reporting an error including a diagnostic, which facilitates its rapid correction.

Reliability and correctness are very important features for many software products, such as financial, medical, transportation, engineering and military applications. In her recent excellent book, *Safeware, System Safety and Computers*, N. G. Leveson[2] states, "Not only must a language be simple, but it must encourage the production of simple and understandable programs. Although careful experimental results are limited, some programming language features have been found to be particularly prone to error --among them pointers, control transfers of various kinds, defaults and implicit type conversions, and global variables. Overloading variable names so that they are not unique and do not have a single purpose is also dangerous. On the other hand, the use of languages with static type checking and the use of guarded commands (ensuring that all possible conditions are accounted for in conditional statements and that each branch is fully specified as to the conditions under which it is taken) seem to help eliminate potential programming errors. Some of the most frequently used languages (such as C) are also those that, according to what is known about language design, are the most error prone." Riehle[3] makes a very strong case for Ada where safety is required.

We wish to raise the heretical opinion that most of the languages previously described as "object oriented" would be more accurately described as class oriented. Ada, in

Dr. Robert C. Leif is vice president and chief scientist of Newport Instruments and its Ada_Med Division. He can be contacted at rleif@rleif.com. Tom Moran is founder and president of Decision Aids. He can be contacted at tmoran@bix.com. Randall L.Brukardt is president and co-founder of R.R. Software Inc. He can be contacted at Randy@rrsoftware.com

common with the other members of the Pascal family of languages, specifies ranges and therefore describes objects in a manner that is common to the scientific and engineering practice of establishing tolerances for objects.

The new decimal types and Ada's very strong emphasis on readability will provide some degree of familiarity for COBOL programmers. As with the other object based programming languages, comment blocks are still required for the actual descriptions that precede the procedures and functions (methods). However, Ada is sufficiently expressive to permit the number of comments for the algorithmic text to be minimized compared to other languages. Information contained in the source text is checked by the compiler; comments are not. The extra information present in well written Ada can be employed by the compiler during code optimization to increase the efficiency of programs.

Description of Ada Syntax

Ada source text is split between a specification, the contract with the outside world, and a body, the implementation. The public part of the specification describes the types, objects, procedures and functions, that can be made visible to the rest of the program. The body, which is not visible to the rest of the program, implements these procedures and functions. Congruence of the body is assured because any change in content to that contained in the previously compiled specification will result in an error when the body is compiled.

Please notice in the bodies of the examples given, that although the arguments of the procedures and functions can be formatted by position, most have the very readable notation (Formal => Actual Parameter). The beginning and end of all of the standard programming constructs are specified and the word end, which terminates all procedures and functions, can be followed as shown in the listings below by the name of the procedure or function. Thus, the compiler is always informed where the programmer thinks it is. In order to facilitate comprehension, the listings below are wordier than most production Ada source text.

Since, software engineers are expensive compared to hardware, it is very cost effective to apportion as much of the checking of program correctness, as possible, to the hardware. In fact, several of the authors originally used Ada on an Intel 8088 IBM PC XT. Comparatively, the present PCs are personal mainframes. The growth in hardware capacity has rendered insignificant the hardware requirements of Ada compilers.

Although Ada was created under the auspices of the United States Defense Department with one of its main applications being embedded systems, such as weapons, Ada is a general purpose programming language. The US Defense Department has 50,000,000 lines of Ada source text. Ada is a proven software engineering tool, which reduces both programming and maintenance costs.

Ada 95[4] consists of a core and nine annexes: Predefined Language Environment, Interfaces to Other Languages, Systems Programming, Real-Time Systems, Distributed Systems, Information Systems, Numerics, Safety and Security, and Obsolescent Features. This permits compiler vendors and, potentially, third parties to supply their customers with the core language and only the relevant annexes. The purchase order form for Windows example described below only employs the Predefined Language Environment, Interfaces to Other Languages, and Information Systems annexes. An Ada program to control a washing machine might only employ the Real-Time Systems Annex.

Purchase Order Example

We will demonstrate the readability and object properties of Ada by presenting and explaining representative parts of the source text of a purchase order program, which runs under Windows 95/NT. This program is constructed from multiple Ada packages. The first step is to model multiple currencies. This will permit customization for the international market.

All Ada reserved words in the examples below and when first described in the text will be lower case bold. The Ada data types and specific symbols will not be shown in bold, but will be explained as they appear. The first step in our example is to create a package which will accurately model multiple currencies including: United States Dollars and Japanese Yen. This starts with the decimal fixed point type. The Ada equivalent of the COBOL picture operations is found in the Ada.Text_IO.Editing package, which is described in the Information Systems Annex. The **with** statement below makes the visible operations and types of this **package** available to the **body** of the PO_Entry package, parts of which are shown in Listing 1.

The word **generic** indicates that the types of the objects beneath it and above the word package have not been completely specified. Instead a template has been created for future use. Ada uses a doubled dash as the start of a comment symbol, "--". Comments extend from the "--" to the end of the line. This eliminates any ambiguity of determining what is and what is not commented out. Virtually all Ada editors include the functionality of adding and removing the "--" from the first two positions of multiple lines.

The PO_Entry package specification contains the data types including the generic to build the various currencies and a description of the two **function**s and one **procedure** required to make the form. The specification only provides the names and types of objects imported and exported by the functions and procedures and the names of the functions and procedures. A specification does not provide any information on the implementation, which is given in the body.

```
-- Comment block.A generic is employed to
-- permit the selection of the appropriate
-- formats for the individual currencies.
-- An incomplete decimal format is used for
-- the Currency and the Currency_Per_Dollar.

-- Yen was used for this example.
-- For a real-world application, the data
-- types that describe a currency would be
```

```
-- stored as fields in a database, such as
-- SAGE-st[13]. The text fields are also
-- included as generic parameters, since they
-- will be replaced for other languages.


generic
   -- In order to force specification of some
   -- items, no defaults are provided.
   -- Currency is an incomplete decimal type;
   -- delta specifies the smallest decimal
   -- fraction; for example, 0.01 for one
   -- cent.
   -- <> means not specified
   -- digits specifies the total number of
   -- digits including those in the decimal
   -- fraction.

   type Currency is delta <> digits <>;
   Currency_Pic_String: String;
   Type Exchange_Rate is delta <> digits <>;
   Currency_Per_Dollar : Exchange_Rate;
   -- This is the ratio of the value to the
   -- US Dollar, which is the reference
   -- currency.
   -- The Currency picture fields below
   -- specify, in a manner similar to COBOL,
   -- how the amounts for each currency will
   -- be displayed and are defaulted to USA
   Currency_Indicator : String := "$";
   Currency_Fill      : Character := '*';
   Currency_Separator : Character := ',';
   Currency_Radix_Mark: Character := '.';

   -- The text below is shown in the purchase
   -- order form.
   -- The default is set to English
   Product_Title : String := “Know-All
      Software System”;
   Support_Title : String := “Support
      contract";
   Word_Tax      : String := "tax";
   Word_Pay      : String := “Please pay”;

package PO_Entry is
   procedure Display_Form;
   function Quit_Request return Boolean;
   -- call this to see if user wants to quit
   procedure Finish;
end package PO_Entry;
```

```
-----------------------------------------
------------------------------
-- Later, in the specification of another
-- package, PO_Form, the generic currency
-- becomes Japanese Yen.

-- We need a Purchase Order Entry package for
-- a specific currency.
-- For this example with the Japanese Yen,
-- this will require whole numbers
--(no fractional yen), a Yen sign instead of
-- a '$', and a yen/dollar conversion rate.

type Yen is delta 1.0 digits 7;
type Yen_Per_Dollar is delta 0.1 digits 4;

--The generic template is now filled in.
package Japan_PO_Entry is new PO_Entry(
   Currency => Yen,
   Currency_Pic_String => "$$_$$$_$$9",
   Exchange_Rate => Yen_Per_Dollar,
   Currency_Per_Dollar =>123.66,
   Currency_Indicator  =>(1 .. 1 =>
      Ada.Characters.Latin_1.Yen_Sign),
   Currency_Separator  =>',',  );
   -- Note that in a real system the exchange
   -- rates would come from an externalsource,
   -- since its value would have to be changed
   -- daily or faster!
   -- We now have a Japanese Yen.
   -- Since the language for the purchase
   -- order form is English, the defaults for
   -- the other fields are used.
```

Listing 1 includes both of the specification of the PO_Entry package, which creates the generic currency, and a part of the specification of another package, PO_Form, which transforms Currency into Yen.

Now that we have successfully modeled different types of currencies, it is possible to use this software for one of the great promises of the Internet, international commerce. In order to complete this task, we must employ another Ada tool, R.R. Software’s Class Library Ada Windows, CLAW[5]. CLAW provides an object-oriented Ada 95 binding to the Win32 Graphical User Interface (GUI) used in Microsoft's Windows NT and Windows 95. CLAW hides the underlying Windows 95 code by providing a thick binding which takes care of the entire conversion from Ada 95 to the Microsoft Windows 95/NT operating system. It uses Ada’s tagged

types (classes) to provide inheritance and dynamic binding.

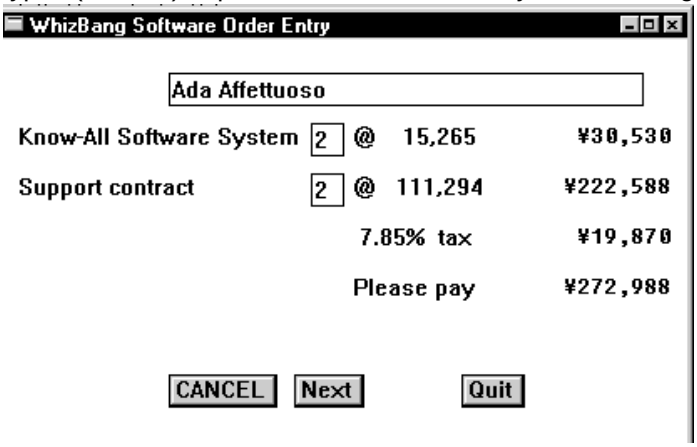

Figure 1 shows a purchase order form with data entry boxes. This form was created with CLAW. There are three data entry boxes. The first is for the customer name, in this case, Ada Affettuoso. The second and third, respectively, contain the numeric values to order copies of Know-All Software Systems and support contracts. The program has calculated the extensions, sale's tax and total. The bottom buttons provide the user with the choices Cancel, Next, and Quit. The internationalization of Ada is demonstrated by having the transaction in Yen

An example of Ada source text from the body of PO_Entry, Procedure Display_Form is shown in Listing 2. The body of PO_Entry **with**s (imports) CLAW and 12 of its children. Each of the three data entry boxes and three push buttons is created by inheriting from a **tagged type** (class) which was previously created in CLAW. In the example below, Cancel_Buttons, for instance, is created from a previous Push_Button type. The term "**null record**" means no new fields were added to the previously existing record.

```
-- Ada shows the inheritance tree by preceding
-- the name of the child, in this case,
-- Push_Button with the name of the parent,
-- Claw, to produce Claw.Push_Button.

type Cancel_Buttons is new
Claw.Push_Button.Button_Type
  with null record;
-- Cancel_Buttons inherits Create and the
-- other procedures and functions.

-- When_Notify is an inherited procedure,
-- which will have to be overridden for this
-- application.

procedure When_Notify(Button : in out
Cancel_Buttons;
      Command : in
         Claw.Control_Command_Code_Type;
      Unknown_Command: in out Boolean);
   -- The formal parameter Button has its
   -- import in and export out capabilities
   -- and type specified.

procedure Display_Form is
begin
   Create(Window  => Display,
      Window_Name => "WhizBang Software Order
         Entry",
      Position => (Left=>20, Right=>475,
                   Top=>20, Bottom=>300));
   Show(Window => Display,
        How => Claw.Show_Startup);
   delay 0.1;  -- seconds. Provide time to
      -- draw the screen.
   Create(Customer_Name_Box,
      Display,Position => (100, 20),
   -- The Size must include the sum of the
   -- border allowance height & width and the
   -- box that fits the number of characters
   -- in the Customer Name)
         Size =>
         Claw.Editbox_IO.Size_Of(Display,
         Character_Count =>
            Customer_Name'Length)+
          Claw.Editbox_IO.Border_Allowance);
   Create(Seat_Count_Box, Display,
          Position => (185, 50));
   Create(Support_Count_Box, Display,
           Position => (185, 80));
   Create(Cancel_Button, Text => "CANCEL " ,
          Parent => Display,
          Position => (100, 200));
   Create(Next_Button, Text => "Next ",
          Parent => Display,
          Position => (175, 200));
   Create(Quit_Button, Text => "Quit ",
          Parent => Display,
          Position => (275, 200));

   delay 0.1; --seconds. Provide time to draw
   -- the screen.
   Reset_Screen;

end Display_Form;
```

Listing 2 demonstrates the functionality of CLAW. Sections of the body of PO_Entry are shown. The procedure When_Notify is shown as it would appear in a package specification.

Finally, we come to the main program, which in Ada is a procedure without parameters. Notice all of the algorithms and most of the types are from the withed (imported) packages. Good Ada software requires building and testing of the parts prior to using them in the main program. It then Displays the Form (Display_Form), enters the User_Input

loop where it waits for data entry and exits when there is a PO_Form.Quit_Request. After the User_Input loop is completed, the PO_Form is finished (PO_Form.Finish). The Ada source text resembles standard English except for the use of compound words held together by underscores instead of multiple words, the loss of appropriate endings to verbs and an occasional the.

```
with Ada.Exceptions, PO_Form, Error_Log;
procedure PO_Demo is
begin
   PO_Form.Display_Form;
   User_Input:
   loop
      exit when PO_Form.Quit_Request;
   end loop User_Input;
   PO_Form.Finish;

exception
   when Unanticipated_Problem : others =>
   -- 'others' means handle any exception not
   -- otherwise handled
   -- Unanticipated_Problem is the name of
   -- the particular exception occurrence
   Error_Log.Log_Exception(
      Ada.Exceptions.Exception_Information
         (Unanticipated_Problem));

end PO_Demo;
```

Listing 3 shows the main program, PO_Demo.

An exception, Unanticipated_Problem, has been included to assist in determining the cause of errors. Storing of a description of the error that raised the exception and a walk-back in the Error.Log file greatly facilitates its transfer to technical support. Subsequently, the information can be read from the file onto a window.

For demonstration purpose, an exception in a procedure File_Away_PO, which is in the body of the PO_Entry package, was forced. An attempt was made to store a 21 character string in 20 character string. As illustrated in Listing 4, the exception handler appended the following information to the file “Error.Log”.

```
CONSTRAINT_ERROR
Index or Subtype out of bounds - Pos of Error
Value = 21
On Line Number 341 In PO_ENTRY.FILE_AWAY_PO
Called from line number 383 In PO_ENTRY
Called from line number 469 In
CLAW.LOW_WND_PROC.PROCESS_CHILD_COMMAND
Called from line number 406 In
CLAW.LOW_WND_PROC.CHECK_MESSAGE
Called from line number 138 In
CLAW.BASIC_WINDOW.BASIC_WND_PROC
Called from line number 166 In
CLAW.LOW_WND_PROC.CONTROL_WND_PROC.CALL_ORI
GINAL_WND_PROC
Called from line number 557 In
CLAW.LOW_WND_PROC.CONTROL_WND_PROC
Called from line number 648 In
CLAW.MESSAGE_LOOP_TYPE
```

Listing 4 shows the run-time error report produced by the R.R. Software Windows 95 Ada compiler when the program receives a 21 instead of a 20 character string. The type of error is identified in the first and second line. This is followed by a list of the locations of all of the parts of program that are involved with the processing this string.

Conclusion

In short, we have demonstrated that the use of the decimal type, generics, and object features of Ada significantly facilitates the development of commercial Windows applications. The advanced exception reporting capability of Ada can be employed to decrease the cost of customer support, while increasing customer satisfaction.

In the near future, a GUI builder[5] that includes drawing and other tools for the creation of GUIs and automatic source text generation will be available. The GUI builder, which is based on the CLAW binding, produces object-oriented, portable Ada source text with user selected, readable identifiers. The CLAW builder also creates resources in the standard resource language. These resources couple to the generated Ada code to help avoid errors. Because CLAW can take advantage of the Ada advanced real-time facilities, it is now possible to create reliable multitasking software for Windows programming.

Although CLAW is an R.R. Software product, since our application requires the Information Systems Annex, we compiled our source text with the GNAT[6] Ada compiler. Software portability and interoperability are an ingrained part of the Ada culture. Any Ada compiler for Windows which has passed the US Government Validation Suite will compile CLAW and permit its use. Conversely, when R.R. Software completes its implementation of the Information Systems Annex, it will compile this program.

Ada has now taken an even greater step towards portability with the implementation of the Ada Semantic Interface Specification. A public interface to the compilation information now exists and is well on its way to ISO standardization. The Ada Semantic Interface Specification facilitates the development by both Ada compiler vendors and third parties of vendor independent debuggers and other sophisticated tools. No other language has even approached this level of portability; and present commercial software marketing practices are inimical to true standardization.

There are several excellent web sites[7-9] and a newsgroup[10] that include: further information about Ada, free Ada software including compilers, lists of books, and tutorials. An Ada compiler that produces J codes and works with the JAVA environment is already available[11, 12].

References:

1. Nicolaisen, N. Letters to the Editor, *Object Magazine*, 6(12) 7-8, 1997.

2. Leveson, N.G. *Safeware, System Safety and Computers*, Addison-Wesley, Reading, MA 412-413, 1995.

3. Riehle, R. “Can Software Be Safe? --An Ada Viewpoint,” *Embedded Systems Programming*, 9 (13) 28-40, 1996.

4. Ada 95 Mapping/Revision Team, “Ada 95 Reference Manual The Language, The Standard Libraries,” Intermetrics, Inc. 733 Concord Ave. Cambridge, MS. 02138; (ISO/IEC 8652:1995). 1995.

5. CLAW, R.R. Software Inc. home page: www.RRSOFTWARE.com

6. GNAT, AdaCore home page:www.gnat.com

7. Ada IC, SW-ENG home page: sw-eng.falls-church.va.us/

8. The SIGAda web site:www.acm.org/sigada/,

9. The Ada home page: www.adahome.com/ This hosts the FAQs, electronic version of useful documentation, tutorials, and information about the utility of Ada

10. Ada News Group, comp.lang.ada list server: INFO-ADA@LISTSERV.NODAK.EDU.

11. Aonix Home Page: www.aonix.com

12. Intermetrics Applet Magic: www.appletmagic.com/appletmagic.html

13. SAGE-st home page://sageftp.inel.gov/sage/homepage.htm