Mathematical Libraries in Ada [1]

Jan Kok
Centre for Mathematics and Computer Science
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

In this paper a description is given of past and present activities at the CWI related to the programming language *Ada*. After a survey of Ada's history and its design aims, more details are given of language features that are of special interest for the implementation of numerical methods and the construction of large scientific libraries. Characteristics of these libraries, enhanced by the use of Ada, are portability, reliability, and reusability. The paper concludes with an outline of future activities which will take place in international co-operation.
KEY WORDS: Ada, high level language, scientific libraries, portability, reliability, reusability.

1. Introduction

At the end of the sixties it was recognized that the production of computer software was not keeping pace with the increase in availability and possibilities of modern computers. Causes of the so-called 'software crisis' were found in the old-fashionedness of programming methods and tools. Independent efforts to provide a new programming language for common use (one of the tools) merely contributed to the crisis, because of the confusion of languages and the incompatibility of systems.

In a programme launched by the US Department of Defense for modernising both tools and methods, one of the results was the definition of a new programming language for common use. This language, called Ada [1], was primarily designed for the production of large portions of maintainable software for real-time applications. Furthermore Ada's use should enhance software characteristics such as readability, modularity, portability, reusability, and reliability.

With respect to *scientific* computation, Ada was intended in the first place for use in those calculations invoked in software for real-time applications. But the readability, modularity, portability, reusability, and reliability, which make programming in large teams possible as well, make Ada also useful for the production of software for large-scale scientific computation on mainframes.

1. Ada is a registered trademark of the US Government (AJPO).

The need for mathematical software in Ada with the same qualities as software for real-time applications is not yet widely recognized by users. It is generally assumed that computer manufacturers have completely satisfied the demand for software for all kinds of computations. Here, many enjoyable case stories of application failures are tacitly ignored [2]. Perhaps it is the rise of a modern language like Ada which now clearly exposes the shortcomings of previous hardware and software provisions. Possibilities not offered by most other languages are user demands for various real precisions, constructs for writing portable and reusable software, and safe interaction between separate library modules.

One may expect that software produced before, in older languages and probably in many versions for all kinds of machines and systems, should be reconsidered for use in Ada, and that re-design of mathematical software in agreement with the Ada design goals might turn out to be very profitable.

An early initiative was taken by the National Physical Laboratory (NPL) in Teddington and the Centrum voor Wiskunde en Informatica (CWI), to investigate both the new possibilities given by the language Ada for the design of mathematical software and for the construction of scientific libraries for large-scale computation, and to offer solutions for the problems encountered. This led to a project by NPL and CWI that was sponsored by the European Community. In this project, which lasted from 1982 to 1983, guidelines were produced for the design of large, modular, scientific libraries in Ada [4].

In the following sections first a summary of Ada's history and design are given, with a discussion of Ada's design goals and some information about Ada's availability. Next, language features are described which open new ways for the implementation of efficient and portable mathematical software for large scientific libraries. Finally, completed and ongoing activities are described.

2. Ada's history

The story of Ada starts in 1974 when Ada's foster parent, the US Department of Defense (DoD), initiated a programme for obtaining more profits from their software budget, in particular concerning software for embedded systems. After an analysis of software costs, with the conclusion that too much was spent on maintenance and conversion, characteristics were gathered for a programming language in common for all types of machines in use, and for all kinds of applications.

When it became clear that no existing language satisfied the requirements, proposals were requested for the development of a new language. From the selection of four proposals for further evaluation, in 1979 one language design was chosen to become the common high-level language. Its name would be *Ada*, in honour of Augusta Ada Lovelace, daughter of Lord Byron, and as the assistant of Charles Babbage one of the first programmers. In 1980 this language was standardized by the DoD, and a revised version was accepted as ANSI standard in 1983 (ANSI: American National Standards Institute). The process for its becoming an ISO standard continues.

One of the characteristics of Ada is that its text should be clear, readable, and easy-to-understand: the possibility of programming errors and typing errors not being detected should be very low. This characteristic is achieved by:

- offering language expressions for:
  - the declaration of (abstract) data types;
  - making packages of sets of data types, operations, and routines;
  - defining library modules, where components of library modules can be packages as well as individual routines, i.e. procedures, functions, or operators;
  - recovery from aborts by the so-called *handling of exceptions*;
  - programming for distributed systems by the so-called *tasking* facility;
- detection of errors at an early stage by:
  - Ada's strong typing rules and scope rules;
  - the small number of error-prone language concepts;
  - several constructs that encourage structured programming;
  - the inhibition of access to abstract data types from outside;
- possibilities for the design of portable programs by means of language concepts for:
  - obtaining hardware or system information through environmental parameters;
  - connecting appropriate hardware provisions to user-defined data types (and also operators, exceptions, and tasks);
  - reusing software through the *generic* concept (see section 3).

The availability of Ada is still very small. The amount of language constructs and their semantic intricacies make the writing of a compiler and run-time system a major software project. Moreover, only systems which correctly process an official suite of about 2000 test programs are allowed to be called *Ada* (compiler validation). And although the language requirements were such that compilers ought to be able to generate efficient object code, not much can be said about this yet, since the first efforts are spent on obtaining *correct* object code, not *fast*. For Ada to become a success, many more compilers for different machines must become available within the next few years.

Meanwhile, the creation of a new language became just one component of the much larger DoD activity for improving software technology. A related project is the construction of *Ada Programming Support Environments* (APSE) as the indispensable working environments for Ada programmers and for the execution of Ada programmes. An APSE should contain besides the Ada compiler and run-time system, the command language, an Ada-directed editor, libraries, verifiers, a debugger, monitors, data bases, etc.

3. Use of Ada for numerical software

During 1982 and 1983 the National Physical Laboratory, Teddington, and the Centrum voor Wiskunde en Informatica were engaged in an investigation of the possibilities of designing large modular scientific libraries in Ada. The project was sponsored by the Commission of the European Community and culminated in the production of a set of guidelines [4] which include recommendations on the ways in which Ada can and might be used in this context.

The language features which might make Ada particularly useful for the redesigning of numerical software and for the construction of coherent scientific libraries, are:

- Several *floating-point types* can be defined. E.g.

  ```
  type REAL is digits 8;
  ```

  (Meaning: a new floating-point type named REAL is made by specifying a minimal number of 8 significant decimal digits.)
  If hardware floating-point types with different mantissa lengths are available, then the hardware type best suitable to the user-defined type will be associated with it. If the user's requirements cannot be met, the program will not run.
- Type definitions and associated operator definitions can be made for all kinds of numerical data structures, e.g.
  VECTOR, MATRIX, COMPLEX, RATIONAL, etc.
- A *package* in Ada is a capsule containing related definitions of types, operators, and procedures. Moreover, the physical structure of data types can be hidden from the user. For example,

  ```
  package RATIONAL_FIELD is
     type RATIONAL is record
                    NUMERATOR : INTEGER;
                    DENOMINATOR : POSITIVE;
                 end record;

     function "+" ( A, B : RATIONAL ) return RATIONAL;

     -- analogous declarations for "-", "*", "/", ...

   end RATIONAL_FIELD;
  ```

  The declarations of this package can be obtained together by mentioning: **use** RATIONAL_FIELD.
  If the inner declaration is replaced by:

```
type RATIONAL is private;
```

this would make the structure of the type RATIONAL hidden.

- Language concepts are available for the construction of large pre-compiled libraries containing as modules packages, but also individual routines. The language rules enable that consistency checks for parameters (in compile time) will always be possible.
- Modules can be parameterized with *types*, *operators* and other subprograms, as well as with individual values and variables. Such modules are called *generic* subprograms or packages, and it allows library packages to be constructed once for a whole class of user-defined types and the operations thereon.
  For example, for sorting values in a one-dimensional array only one generic sort procedure need be written:

```
generic
  -- ( with three generic parameters:
    type EL_TYPE is private;
    type AR_TYPE is array ( INTEGER range <> ) of EL_TYPE;
    with function "<" ( A, B : EL_TYPE ) return BOOLEAN
        is <>;
  -- )
procedure GEN_SORT ( X : in out AR_TYPE );
```

  This generic procedure is a template for sorting procedures. A concrete sorting procedure can be obtained by substituting a linear array type, its component type, and the operator "<" acting on values of the component type. The specification of such a procedure would be

```
procedure SORT_INTEGERS ( X : in out INTEGER_ARRAY );
```

- *Exceptions* and the recovery from raised exceptions can be used for the description of all abnormal events. E.g., when the declaration of an exception SINGULAR_MATRIX is available, a routine can execute the statement

```
raise SINGULAR_MATRIX;
```

  to cause abnormal termination of the called routine. However, the user can catch raised exceptions in so-called *exception handlers,* in order to execute some recovery action.

- The concept of *tasks* is a high-level concept for describing in a readable and reliable way concurrent activities and the safe communication between tasks, and between a user and his tasks. Tasks can be used to describe in a clear way the provision of numerical services to processes in embedded systems, but also the processing of distributed computations for special architectures, especially for general-purpose multi-processors.

A conclusion of the NPL/CWI investigation was that Ada is a useful language and several Ada features are of much interest for the redesigning and construction of large scientific libraries. Difficulties caused by idiosyncracies of the language syntax and semantics can be overcome satisfactorily in most of the cases, and solutions were offered. Moreover, since elementary mathematical provisions in Ada style were expected to be indispensable for Ada programmers, a standard definition and the early implementation of basic functions packages were emphatically recommended in [3].

In [4] also examples of library components were given, and language items were listed which ought to be cleared (by Ada implementors) or preferably changed in a future language revision.

For more information and details one is referred to the complete report [4].

4. Future activities

Members of the CWI Numerical Mathematics Department are currently co-operating with colleagues from six European Community countries (and Sweden) in the EC-supported *Ada-Europe Numerics Working Group.* This group has already produced several documents on all aspects of the implementation of numerical libraries in Ada and the design of new methods. Co-operation is sought with a US counterpart.

A problem the group is facing is that the need for newly-designed mathematical software with properties that Ada can provide (reliable, well-designed, readable, and portable modules in coherent packages) is not generally recognized. Ada offers the possibility of imposing a hierarchical structure onto libraries, through encapsulation of related declarations. Furthermore, consistency checks of passed parameters are maintained throughout the library's life cycle. Finally, relying on the mathematical provisions made available by a computer manufacturer has sometimes been a bad experience. It is regrettable that the very experts on the analysis of the results of long computations have been ignored so often.

Meanwhile, co-operation between the CWI and the Numerical Algorithms Group (NAG, Oxford) and NPL (among others) has been established, with the final aim of making all common numerical provisions available to Ada programmers. As a start, pilot implementations will be made of basic modules of numerical libraries in Ada, for which subsidies have been granted.

REFERENCES

1. ANSI/MIL-STD 1815 A. (1983). *Reference Manual for the Ada Programming Language.*
2. R.L. BABER (1984). Software development: science or patchwork? *CWI Newsletter 2,* 18-34.
3. J. KOK, G.T. SYMM (1984). A proposal for standard basic functions in Ada. *Ada Letters Vol. IV.3,* 44-52.
4. G.T. SYMM, B.A. WICHMANN, J. KOK, D.T. WINTER (1984). *Guidelines for the Design of Large Modular Scientific Libraries in Ada,* NPL Report DITC 37/84, CWI Report NM-N8401.