https://ntrs.nasa.gov/search.jsp?R=19920002446 2020-03-17T15:18:00+00:00Z

IN-61-CR
46428
p. 14

Deploying Expert Systems in Ada

(NASA-CR-188938) DEPLOYING EXPERT SYSTEMS IN Ada (Research Inst. for Advanced Computer Science) 14 p CSCL 09B
N92-11664
Unclas
G3/61 0046428

S. Daniel Lee
Bradley P. Allen

Inference Corporation

October 1989

Cooperative Agreement NCC 9-16
Research Activity No. SE.19

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

ricis

Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Deploying Expert Systems in Ada

[illegible]

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Inference Corporation. Dr. Charles McKay served as RICIS research coordinator.

Funding has been provided by the Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Robert T. Savely, of the Software Technology Branch, Information Technology Division, Information Systems Directorate, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

Deploying Expert Systems in Ada*

S. Daniel Lee and Bradley P. Allen

Inference Corporation
5300 W. Century Blvd.
Los Angeles, CA 90045

Abstract

As the Department of Defense Ada mandate begins to be enforced actively, interest in deploying expert systems in Ada has increased. This paper introduces a prototype Ada-based expert system tool called ART/Ada. This prototype was built to support research into the language and operational issues of expert systems in Ada. ART/Ada allows applications of a conventional expert system tool called ART-IM (Automated Reasoning Tool for Information Management) to be deployed in various Ada environments with efficient use of time and space. ART-IM, a C-based expert system tool, is used to generate Ada source code which is compiled and linked with an Ada-based inference engine to produce an Ada executable image. The future research directions call for improved support for real-time embedded and distributed expert systems. ART/Ada will be used to implement several prototype expert systems for the Space Station Freedom Program testbeds.

1. Introduction

1.1. Motivation

As the Department of Defense mandate to standardize on Ada as the language for embedded software systems development begins to be actively enforced, interest from developers of large-scale Ada systems in making expert systems technology readily available in Ada environments has increased.

Two examples of Ada applications that can benefit from the use of expert systems are monitoring and control systems and decision support systems. Monitoring and control systems demand real-time performance, small execution images, tight integration with other applications, and predictable demands on processor resources; decision support systems have somewhat less stringent requirements. An example project that exhibits the need for both of these types of systems is NASA's Space Station Freedom. Monitoring and control systems that will perform fault detection, isolation and reconfiguration for various on-board systems are expected to be developed and deployed on the station either in its initial operating configuration or as the station evolves; decision support systems that will provide assistance in activities such as crew-time scheduling and failure mode analysis are also under consideration. These systems will be expected to run reliably on a standard data processor, currently envisioned to be a 4-16 megabyte 80386-based workstation. The Station is typical of the large Ada software development projects that will require expert systems in the 1990's.

Another large-scale application that can be benefited from the Ada-based expert system tool technology is the Pilot's Associate (PA) expert system project for military combat aircraft [9]. Funded by the Defense Advanced Research Projects Agency (DARPA) as part of its Strategic Computing Program, the PA project attempts to automate the cockpit of military combat aircraft using Artificial Intelligence (AI) techniques. A Lisp-based expert system tool, ART (Automated Reasoning Tool), was used to implement the Phase I

*This paper will be published in the proceedings of the TRI-Ada '89 Conference to be held during October 22-26, 1989 in Pittsburgh, PA.

prototype. An Ada-based expert system tool can provide a migration path to deploy the prototype on an on-board computer because Ada cross-compilers are readily available to run Ada programs on most embedded processors used for avionics.

1.2. Approach

Inference Corporation developed an expert system tool called ART (Automated Reasoning Tool) that has been commercially available for several years [11]. ART is written in Common Lisp and it supports various reasoning facilities such as rules, objects, truth maintenance, hypothetical reasoning and object-oriented programming. Last year, Inference introduced another expert system tool called ART-IM (Automated Reasoning Tool for Information Management), which is also commercially available [12]. ART-IM is written in C and it supports a major subset of ART's reasoning facilities including rules, objects, truth maintenance and object-oriented programming. Both ART and ART-IM have been successfully used to develop many applications which are in daily use today [5], [16], [17].

Our approach in designing a prototype Ada-based expert system tool was to use the architecture of proven expert system tools: ART and ART-IM. ART-IM was selected as a baseline system because C is much closer to Ada. While ART-IM's inference engine was reimplemented in Ada, ART-IM's front-end (its parser/analyzer and user interface) was not. Instead, ART-IM was enhanced to generate Ada source code that would be used to initialize Ada data structures equivalent to ART-IM's internal C data structures. This approach allows the user to take full advantage of ART-IM's interactive development environment while developing an application; once the development is complete, the application is converted to Ada source code that is compiled and linked with the Ada runtime kernel.

1.3. The ART-IM Expert System Tool

ART-IM is a general purpose expert system tool written in C. It consists of

- a runtime kernel,
- a C deployment compiler, and
- an interactive development environment.

ART-IM's kernel supports the following features:

- a forward-chaining rule system based on the Rete algorithm [6],
- an object system,
- object-oriented programming,
- a justification-based truth maintenance system(JTMS), and
- explanation generation utilities.

ART-IM supports deployment of applications in C using a C deployment compiler that converts an application into C data structure definitions in the form of either C source code or object code.

ART-IM's interactive development environment includes a highly functional user interface that allows browsing and debugging of the knowledge base and an integrated editor that allows incremental compilation.

ART-IM is available for VMS, MVS and MS-DOS environments.

2. ART/Ada: An Ada-based Expert System Tool

2.1. Ada Runtime Kernel

The ART/Ada runtime kernel is composed of the following components:

- an inference engine,
- a procedural interface package,
- a memory management package, and
- Ada deployment compiler utilities.

ART/Ada's inference engine is an Ada implementation of ART-IM's inference engine and is functionally identical to ART-IM's.

ART/Ada's procedural interface includes all public functions in ART-IM except for those that are used only during the development phase and those that are part of ART-IM's user interface toolkit. ART/Ada's procedural interface can be used either in the right-hand side of a rule, or directly in user's Ada programs. The procedural interface includes data type conversions between the Ada data types and the ART-IM data types, predicates, operations on ART-IM objects, ART-IM commands, I/O functions and math functions.

ART/Ada's memory management package uses the Ada features **new** and **unchecked_deallocation** to allocate and deallocate memory.

The ART/Ada runtime kernel contains utilities called by the Ada code generated by the Ada deployment compiler.

2.2. Ada Deployment Compiler

ART-IM was augmented with an Ada deployment compiler to support ART/Ada. As shown in figure 2-1, its input is an ART-IM source file, and its output is Ada source files. At any point after an ART-IM source file is loaded into ART-IM, and the knowledge base is initialized for execution, the Ada deployment compiler may be invoked to generate the Ada source code that would initialize the internal data structures of ART/Ada. An Ada package specification generated by ART-IM for an example application called MY_EXPERT_SYSTEM is as follows:

```
-- generated automatically by ART-IM
package MY_EXPERT_SYSTEM is

  -- initialize the application.
  procedure INIT;

end MY_EXPERT_SYSTEM;
```

An Ada main program that the user would write to initialize and run the application would look like this:

```
-- This is a main program written by the user
-- ART is a public package of ART/Ada
with ART, MY_EXPERT_SYSTEM;
procedure MAIN is
begin
  MY_EXPERT_SYSTEM.INIT;    -- initialize it
  ART.RUN(-1);              -- run it.
end MAIN;
```

In addition to generating Ada source code that represents the knowledge base, the Ada deployment compiler also generates a call-out interface module that is used to call Ada subprograms from ART/Ada. ART-IM provides a language to specify the call-out interface for calling Ada subprograms from ART-IM or ART/Ada. The Ada deployment compiler is written in C and is linked with ART-IM.

2.3. Ada Call-in and Call-out

It is common that an expert system application calls out to a procedural language such as Ada from an expert system shell. Since ART-IM is used to develop an ART/Ada application, it is critical to allow the user to call out to Ada from ART-IM.

An expert system application often calls public functions of an expert system tool from a procedural language (e.g. Ada). Since ART-IM is written in C, each public function must be provided with an Ada binding to be called from Ada.

A consistent Ada call-in and call-out interface is provided for both development and deployment environments so that the user-written Ada code runs without modification when it is deployed in Ada after being developed in ART-IM. The ART-IM Ada binding consists of Ada functions that call ART-IM's public functions written in C. The specification of public functions in both the ART-IM Ada binding and the ART/Ada runtime kernel is identical.

Not all Ada compilers support the feature of calling Ada from C. On VAX/VMS, the DEC Ada compiler can be used because both DEC C and DEC Ada compilers confirm to the VMS calling standard. On Unix platforms, the Verdix Ada compiler can be used because it supports this feature well; it is already being used as an in-house development tool for the ART/Ada project; and it is used by many Ada programmers on Unix platforms. This restriction exists only on an ART-IM development platform and does not prevent the users from porting generated Ada code and the ART/Ada runtime kernel to other Ada compilers and hardware platforms. In fact, ART/Ada has been already ported to multiple Ada compilers including DEC, Alsys and Verdix and multiple hardware platforms such as a VAX/VMS, a Sun and an IBM PS/2.

Ada data types supported for the call-in and call-out interfaces are: 32 bit integer (INTEGER_TYPE), 64 bit float (FLOAT_TYPE), boolean (BOOLEAN_TYPE), string (STRING), and an abstract data type for objects in ART-IM (ART_OBJECT). Table 2-1 summarizes the mapping between ART-IM, C and Ada data types.

ART-IM	C	Ada
integer	long	INTEGER_TYPE
float	double	FLOAT_TYPE
boolean	long	BOOLEAN_TYPE
string	char *	STRING
symbol	char *	STRING
art-object	struct *	ART_OBJECT

Table 2-1: Data Types for Ada Call-in/Call-out

For example, in order to call out to an Ada function, CALC_AVG, using an ART-IM function, calculate-average, define the following in ART-IM[*]:

```
;;; define a function, calculate-average.

(def-user-fun calculate-average
;; Ada name
   :epname    "CALC_AVG"
;; argument types
   :args      ((num1 :float)
               (num2 :float)
               (num3 :float))
;; return type
   :returns   :float
;; Ada compiler
   :compiler :dec-ada)
```

A specification of an Ada package called USER should be also defined as follows:

```
-- ART is a public package of ART/Ada
with ART;
use  ART;
-- USER is a package for user's Ada code
package USER is

  -- Ada function is called, CALC_AVG
  function CALC_AVG(NUM1 : FLOAT_TYPE,
                    NUM2 : FLOAT_TYPE,
                    NUM3 : FLOAT_TYPE)
     return FLOAT_TYPE;

end USER;
```

This Ada function, CALC_AVG, can be called from ART-IM as follows:

```
;;; call an Ada function, CALC_AVG,
;;; to calculate an average.

(calculate-average 50.0 45.0 55.0)
```

which would return 50.0.

2.4. Deployment in Ada

The methodology for developing an ART/Ada application defines three distinct platforms, some or all of which may be the same:

- an ART-IM development platform with Ada call-in and call-out capability on which an application is actually developed and debugged;
- an Ada compiler platform on which either a self-target compiler or a cross-compiler is used to compile Ada source code; and
- a target platform on which an Ada executable image will be deployed.

The development phase would involve the development of an ART-IM program with Ada code that interfaces with ART-IM through an Ada call-in and call-out interface, which occurs on the ART-IM development platform.

[*] The syntax of ART-IM's procedural language is similar to Common Lisp.

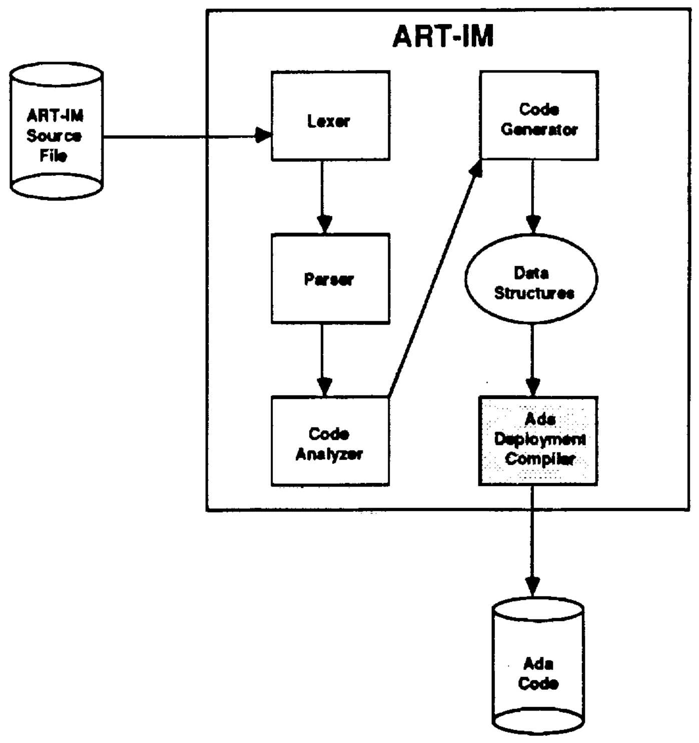

Figure 2-1: Ada Deployment Compiler

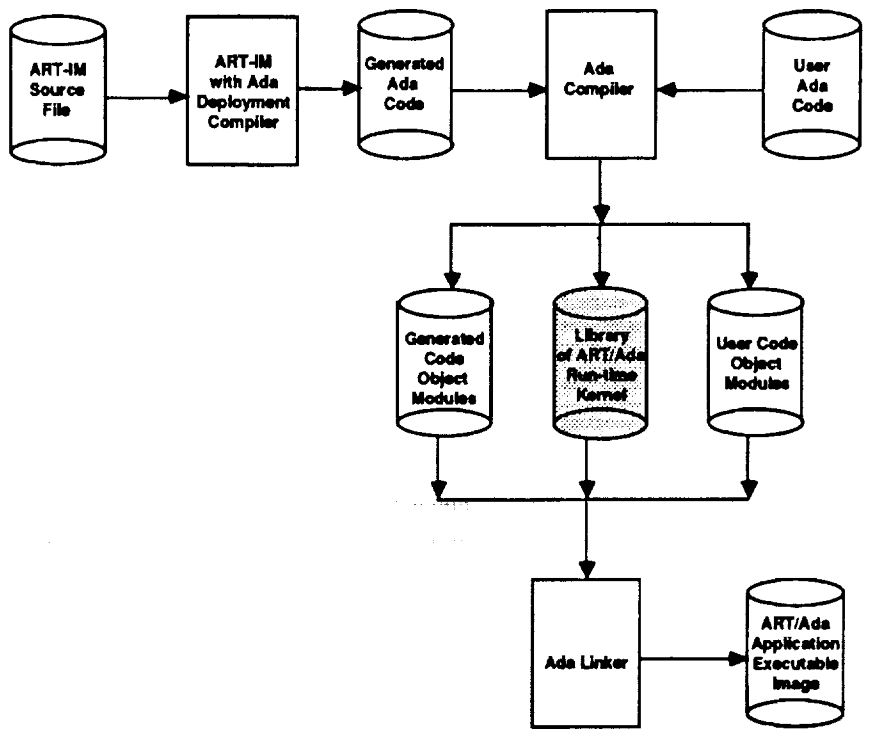
ART-IM Source File
ART-IM with Ada Deployment Compiler
Generated Ada Code
Ada Compiler
User Ada Code
Generated Code Object Modules
Library of ART/Ada Run-time Kernel
User Code Object Modules
Ada Linker
ART/Ada Application Executable Image

Figure 2-2: Ada Deployment Process

The deployment phase would involve compilation of Ada code generated by the ART-IM Ada deployment compiler and Ada code written by the user, which occurs on the platform where the Ada compiler runs. The ART/Ada runtime kernel is provided either in an Ada source code form or as an Ada library that is created using the same Ada compiler. If the Ada compiler is a self-target compiler, the Ada executable image will be deployed on the same platform where the Ada compiler runs. If it is a cross-compiler, it will be deployed on the target platform (which may be an on-board computer).

As shown in Figure 2-2, the following steps are needed to deploy an ART-IM application in Ada:

1. Develop and debug an application using ART-IM's interactive development environment. If necessary, call out to Ada from ART-IM using the standard call-out interface, or call into ART-IM from Ada using the Ada binding.

2. Generate the Ada code from ART-IM using the Ada deployment compiler. This Ada code is portable to any self-host or cross-compiling Ada compilers. If the Ada compiler platform is different from the ART-IM development platform, the generated Ada code can be moved to the platform on which the Ada compiler runs.

3. Compile the generated Ada code and the user-written Ada code using either a self-target compiler or a cross-compiler into an appropriate Ada library of the ART/Ada runtime kernel.

4. Create an Ada executable image by linking an Ada main program.

5. Deploy the Ada executable image on a host computer or on a target system.

3. Discussion

3.1. Performance

The ART/Ada project succeeded in proving that applications of a conventional expert system tool could be deployed in various Ada environments with efficient use of time and space. The preliminary benchmark result of the ART/Ada prototype shows that the speed and the size of ART/Ada prototype is comparable to other tools including C-based tools, although it is somewhat slower and larger than ART-IM.

The address space limitation of current generation embedded processors, such as the MIL-STD-1750A, is 1 megaword (2 megabytes), within which all software systems including the operating system have to run. This might be too restrictive for large expert system applications. New generation embedded processors such as the 80386 would be more than adequate for expert systems developed using ART/Ada.

While Ada compilers are improving, they still have not reached the maturity of C compilers. It has also been observed that both the speed and the size of ART/Ada varies up to 30% depending on which Ada compiler is used. A recent paper discusses the key technical issues involved in producing high-quality Ada compilers [7]. As Ada compiler technology advances, ART/Ada's performance will improve; we expect to narrow the performance gap between ART-IM and ART/Ada.

3.2. Ada Limitations

During the reimplementation of the ART-IM runtime kernel in Ada, several issues concerning the limitation of Ada language arose.

In order to achieve maximum time and space efficiency, ART-IM has been optimized in ways that are not portable to Ada. For example, the type cast feature of the C language has been used both to optimize data structure and to implement an internal memory manager. ART-IM's memory manager maintains its own free list and handles all allocation and deallocation requests from the ART-IM kernel; it allocates large blocks of memory from the system, and then fulfills individual (relatively small) requests for storage from the large blocks. As storage is released, it is added to an internally maintained free list; the blocks themselves are never released back to the system. There are several advantages to this approach: the free space is managed in a common pool by a single facility and is available for allocation of arbitrary data types by using the type cast capability in C; and it is faster than using system routines for small requests. The success of ART-IM's use of type casting relies on other features of the C language definition: there is a direct correspondence between addresses and pointer types; the mapping between data types, including structures and arrays, is well defined and straightforward.

Ada does provide a facility for converting between data types, although this feature has intentionally been made difficult to use. In order to convert from one data type to another, the generic function unchecked_conversion must be instantiated for each conversion required. The existence of a type cast capability in Ada is insufficient to implement the ART-IM features described above, however. No correspondence is guaranteed between the type SYSTEM.ADDRESS and Ada access types. Indeed, on some implementations the underlying representation is different for addresses and access types. The constraint checking requirements of Ada require that the representation of many objects include descriptor information. The format of these descriptors is not defined by the language. Hence, it is impossible to implement the ART-IM style memory manager in Ada using unchecked_conversion. Compared to ART-IM, this has resulted in some loss of efficiency in ART/Ada that allocates and deallocates memory for each data type directly from or to the Ada runtime system using Ada features, new and unchecked_deallocation.

We also discovered other limitations in Ada that do not exist in C:

- ART-IM has an interpreter (similar to a Lisp interpreter) that calls a C function using a C function pointer. To emulate ART-IM's function call mechanism the Ada deployment compiler automatically generates Ada source code for a procedure called FUNCALL that has a large case statement. This case statement contains all the Ada subprograms that are called from an ART/Ada application. Each subprogram is assigned with an ID number. To call an Ada subprogram, the procedure FUNCALL is called with a subprogram ID number. While it may cause maintenance problems, the use of function pointers can provide better performance than the use of the Ada case statement.
- Bit operations (e.g. bitwise exclusive OR, bitwise shift operations, etc.) that may be used to implement efficient hashing algorithms are not provided in Ada. They may be implemented in Ada but only with poor performance.
- Because a math library, which is part of the standard C language, is not part of the standard Ada, it is hard to write portable Ada code that uses math functions.
- Representation specification is not portable because each Ada compiler and/or hardware platform may use a different memory boundary.
- Variant record is the only Ada data type that can be used to implement C's union, but it is not as efficient nor flexible.
- Some Ada compilers do not allow calling an Ada program from another language because Ada is a runtime environment as well as a programming language. When it is supported, many restrictions are usually imposed: the main program must be an Ada program, and exceptions and tasking may not be used by the Ada program called from another language.
- In C, conditional compilation facilitated by preprocessor directives (e.g #define and #if) allows maintaining a single source file for multiple platforms. In Ada, no such facility exists, and multiple files may have to be maintained for multiple platforms.
- An Ada library system may lead to wasted disk space. For example, an Ada library management system requires duplication of the whole library when the body of a package in the library has dual definitions. In C, when functions are defined more than once, they can be simply stored in multiple local libraries while the rest of the program is stored in a main library without duplication. Only one of these multiple local libraries is linked with the

main library.

- A C-style formatting function (e.g. printf, sprintf, etc) is hard, if not impossible, to implement in Ada because the data types of its function arguments are not pre-determined.

Various Ada language issues are being studied by several working groups including the Ada Language Issues Working Group (ALIWG) and the Ada Runtime Environment Working Group (ARTEWG), and will be proposed for the Ada 9X standard [1], [2]. We believe that some of the issues discussed in this paper should also be considered for the Ada 9X.

3.3. Related Work

FLAC (Ford Lisp-Ada Connection) uses a Lisp environment to develop an expert system application and generates Ada code to be deployed in Ada environments [13]. Its knowledge base is specified using a graphical representation similar to that of VLSI design (e.g. OR gates and AND gates). FLAC is similar to ART/Ada because its development environment is not implemented in Ada but Ada deployment is supported. The difference is, however, that FLAC's development environment is based on Lisp, while ART/Ada uses that of ART-IM which is written in C. C and Ada development environments coexist on the same hardware platforms more often than Lisp and Ada development environments do. FLAC, for example, uses a special-purpose Lisp machine for the front-end, and a VAX for the Ada deployment. Both ART-IM and ART/Ada can run on the same hardware. Another difference is that FLAC's input is graphics-oriented while ART/Ada is language-oriented. FLAC's knowledge base is pre-compiled and static, which means that objects may not be added or deleted dynamically at runtime although their values may be changed. This impose major restrictions on the reasoning capability which do not exist in ART-IM and ART/Ada.

CHRONOS is a commercial expert system tool written in Ada that was introduced recently. It is developed and marketed by a French company, Euristic Systems. As its name implies, it supports temporal reasoning capabilities by time-stamping each fact with temporal attributes. Currently, little is published about this tool.

Another commercial tool is an object-oriented programming environment called Classic-Ada [18]. It seems to have its roots in Smalltalk, Flavors and CLOS (Common Lisp Object System). Although Classic-Ada does not support rules, its object-oriented programming features are similar to ART-IM's object system.*

It is reported that several logic-based tools support Prolog in Ada [4], [3], [10]. Although Prolog can be used to implement expert systems, its approach and scope are significantly different from expert system tools such as ART-IM. These tools, therefore, are not covered in this paper.

3.4. Future Work

ART/Ada will be used by several NASA sites to implement prototype expert systems for the Space Station Freedom Program testbeds. This will allow research to understand the potential uses and operational issues of ART/Ada.

Our future research effort will be focused on real-time embedded and distributed applications:

- to meet real-time requirements,

- to support distributed environments (e.g. parallel processors), and

- to fit into embedded processors.

Real-time requirements are still not very well understood [19]. Support for real-time applications in an expert system tool is usually focused on temporal reasoning capability or on better performance [14]. No tool presently available seems to address guaranteed response time. While it is not clear how an expert system tool can satisfy *hard* real-time requirements by guaranteeing response time, its performance could be optimized to satisfy *soft* real-time requirements [15]. Although it is possible to implement temporal reasoning in ART/Ada using existing features, it would be straightforward to build

*When rules are not used, ART-IM can be viewed as an object-oriented programming environment.

temporal reasoning capability directly into ART/Ada.

One way to support parallelism in a Rete-based expert system tool is to parallelize the Rete network [8]. This approach may require specialized hardware. Another approach is a message-passing architecture that allows multiple expert systems to communicate asynchronously. This approach can be implemented by developing multiple ART/Ada programs and a communications package outside of ART/Ada. Ideally, though, a built-in capability should be provided to support multiple cooperating expert systems that can run as multiple processes on a single processor or as distributed processes on multiple processors. If multiple "knowledge-base packages" are supported in a single program, each package can be deployed as an expert system module that would communicate asynchronously with other modules through a message passing mechanism that may have to be customized for each software/hardware platform. Ada tasking would be ideal for implementing this because it is portable and does not require customization.

Although semiconductor technology is improving very rapidly in the commercial sector, embedded processors are still based on the old technology. Modern operating system features such as virtual memory are not readily available on most on-board computers. The resource requirements on these computers such as processor speed and real memory are quite stringent. It is essential that ART/Ada meet these requirements for the emerging new-generation embedded processors such as the Intel 80386, the Intel 80960, and the MIPS RISC chip.

4. Acknowledgments

The authors wish to acknowledge the guidance and support of Chris Culbert, Bob Savely and Bob Brown of NASA Johnson Space Center Mission Planning and Analysis Division, Greg Swietek of NASA Headquarters Space Station Office, and Captain Mark Gersh of USAF Technology and Requirements Planning. Mark Auburn, Don Pilipovich and Mark Wright of Inference Corporation contributed to the project.

References

1. Ada Language Issues Working Group. "Ada Language Issues Working Group (ALIWG) Minutes of 17 August 1988". *Ada Letters IX*, 1 (January February 1989).

2. Ada Runtime Environment Working Group. "Activities of the Ada Runtime Environment Working Group". *Ada Letters IX*, 5 (July August 1989).

3. Bobbie, P.O. ADA-PROLOG: An Ada System for Parallel Interpretation of Prolog Programs. Proceedings of the third Annual Conference on Artificial Intelligence and Ada, 1987.

4. Burback, R. PROVER: A First-order Logic System in Ada. Proceedings of the third Annual Conference on Artificial Intelligence and Ada, 1987.

5. Dzierzanowski, J.M. et. al. The Authorizer's Assistant: A Knowledge-based Credit Authorization System for American Express. Proceedings of the Conference on Innovative Applications of Artificial Intelligence, AAAI, 1989.

6. Forgy, C.L. "RETE: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem". *Artificial Intelligence 19* (1982).

7. Ganapathi, M., Mendal, G.O. "Issues in Ada Compiler Technology". *Computer 22*, 2 (February 1989).

8. Gupta, A. *Parallelism in Production Systems*. Pitman Publishing, 1988.

9. Hugh, D.A. "The Future of Flying". *AI Expert 3*, 1 (January 1988).

10. Ice, S., et. al. Raising ALLAN: Ada Logic-Based Language. Proceedings of the third Annual Conference on Artificial Intelligence and Ada, 1987.

11. Inference Corporation. *ART Version 3.2 Reference Manual*. Inference Corporation, 1988.

12. Inference Corporation. *ART-IM 1.5 Reference Manual*. Inference Corporation, 1989.

13. Jaworski, A., LaVallee, D., Zoch, D. A Lisp-Ada Connection for Expert System Development. Proceedings of the third Annual Conference on Artificial Intelligence and Ada, 1987.

14. Laffey, T.J., Cox, P.A., Schmidt, J.L., Kao, S.M., Read, J.Y. "Real-Time Knowledge-Based Systems". *AI Magazine 9*, 1 (Spring 1988).

15. Laffey, T. S. Weitzenkamp, Read, J., Kao, S., Schmidt, J. Intelligent Real-Time Monitoring. Proceedings of the National Conference on Artificial Intelligence, AAAI, 1988.

16. Nakashima, Y. Baba, T. OHCS: Hydraulic Circuit Design Assistant. Proceedings of the Conference on Innovative Applications of Artificial Intelligence, AAAI, 1989.

17. O'Brien, J. et. al. The Ford Motor Company Direct Labor Management System. Proceedings of the Conference on Innovative Applications of Artificial Intelligence, AAAI, 1989.

18. Software Productivity Solutions, Inc. *Classic-Ada User Manual.* Software Productivity Solutions, Inc, 1988.

19. Stankovic, J. A. "Misconceptions about Real-Time Computing". *Computer 21*, 10 (October 1988).