ART/Ada Design Project - Phase I Task 1 Report: Overall Design

Status Report

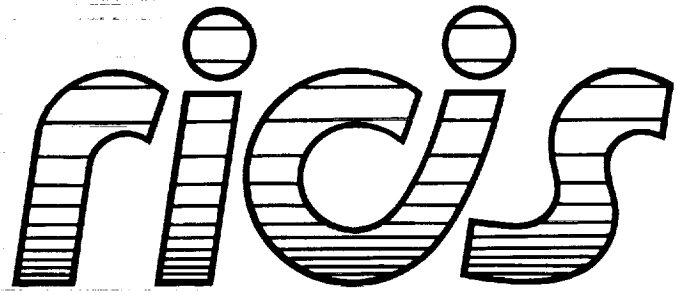
Bradley P. Allen

Inference Corporation

October 24, 1988

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

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



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.

ART/Ada Design Project - Phase I
Task 1 Report: Overall Design

Status Report

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.

ART/Ada Design Project - Phase I

Task 1 Report: Overall Design

Status Report

for

Subcontract 015

RICIS Research Activity SE.19

NASA Cooperative Agreement NCC-9-16

March 1988 - October 1988

Bradley P. Allen

Inference Corporation

5300 W. Century Blvd.

Los Angeles, CA 90045

24 October 88 16:47

Table of Contents

List of Figures

1. Introduction

Both the Ada programming language and expert systems technology have been described as ways to increase software productivity and reliability; given this, it is natural to expect that making expert systems technology readily available in Ada development environments would be an worthwhile objective. As the Department of Defense mandate to standardize on Ada as the language for embedded software systems development begins to be actively enforced, interest on the part of developers of large-scale Ada systems in achieving this goal 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 limited demands on processor resources; decision support systems have somewhat less stringent requirements. An example project which exhibits the need for both of these types of systems is NASA's Space Station. 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 as a 1-4 MIPS, 1-4 megabyte workstation. The Station is typical of the large Ada software development projects that will require expert systems in the 1990's.

The question that we wish to address is: can expert system building tools (ESBTs) be built to support the development of Ada-based expert systems for projects of this type and magnitude? There is evidence that this is the case:

- commercial successes in the development and deployment of expert systems for both decision support and monitoring and control applications,
- existing ESBTs implemented in widely-supported compiled procedural languages (such as C), and
- initial work on Ada-based ESBTs.

Work on Ada-based ESBTs has involved the implementation of prototypes supporting knowledge representations such as production systems, frames, and logic. Implementation approaches that have been explored to date include Ada implementations of expert system interpreters and automated translation of specific expert systems into Ada source code.

Preliminary results are encouraging, but there is still a long way to go before commercial-quality ESBTs

can be provided. Problems that have been encountered in prototype efforts are the size of the executable image, the size of generated code (in translation-based approaches), and overhead due to run-time interpretation of variant datatypes. This work has also exposed several design tradeoffs whose resolution is poorly understood:

- compilation vs. interpretation of the knowledge base,
- strong vs. "weak" typing of the expert system data types,
- minimal vs. extensive use of tasking,
- Ada-based vs. "traditional" expert systems language syntax, and
- a complete Ada expert system development environment vs. an existing non-Ada development environment with support for deployment in Ada.

The difficulty of making good decisions among the design tradeoffs is compounded by the fact that the current state of Ada compiler technology tends to make the ESBT's performance (as well as expert systems implemented using it) overly dependent on the choice of compiler. More work needs to be done to allow the design and implementation of an Ada-based ESBT whose performance characteristics are predictable across platforms. Beyond performance considerations, Ada-based ESBTs also need to address these methodological issues:

- conformance with standard Ada development methodologies in the development of the ESBT,
- verification and validation of the ESBT, and
- support in the ESBT for the validation and maintenance of expert systems.

Finally, there are general questions that must be addressed by the Ada and expert systems communities at large:

- hardware-level support for concurrent tasks,
- the integration of expert systems development methodologies (and the use of ESBTs) into standard Ada development methodologies, and
- methods for the design and development of reusable knowledge bases.

Under subcontract to University of Houston - Clear Lake as part of the Cooperative Agreement between UHCL and NASA Johnson Space Center, Inference Corporation is conducting an Ada-Based ESBT Design Research Project. The goal of the research project is to investigate these issues in the context of the design of an Ada-based expert systems building tool. We are taking the following approach: using an existing successful design as a starting point, analyze the impact of the Ada language and Ada development methodologies on that design, redesign the system in Ada, and analyze its performance using both complexity-theoretic and empirical techniques. The research project will attempt to achieve a

comprehensive understanding of the potential for embedding expert systems in Ada systems, for eventual application in future projects.

This research project consists of four discrete tasks and reports analyzing the research results at the end of each task. If the research demonstrates feasibility of the redesign, the design effort will continue into a second phase that will determine the feasibility of the redesign of a larger subset.

In Task 1, we began our effort with an exploration and comparision of design alternatives. Because the research objective is the demonstration of the feasibility of developing expert systems systems in Ada, and because the ART program is an existing example of a development tool for constructing expert systems systems, ART will serve as a baseline for the design. This report identifies an architecture for an initial Ada version of ART, henceforth referred to as *ART/Ada*, and addresses the following topics:

- The methodology followed in the design and implementation of the architecture.
- The design alternatives and the rationale for selecting among them.
- Differences in the Ada architecture which may limit or expand functionality.
- Language differences which require or allow differentiation between the base software and the Ada version.

The report will serve as a framework for the further refinement of algorithms entailed by a selected subset of the Ada version of ART, and constitutes the deliverable for Task 1.

In Task 2, we will begin to further refine the algorithms specified in the overall design, resolving and documenting any open design issues, identifying each system module, documenting the internal architecture and control logic, and describing the primary data structures involved in the module. This data will be compiled and provided in a deliverable report, which will contain Ada specifications of exported packages.

In Task 3, we will develop a comprehensive protocol for the analytic and empirical analysis of the performance characteristics of the specified algorithms, and implement in Ada the algorithms necessary to support the analytic and empirical analysis. This protocol will be compiled and provided in a deliverable report.

In Task 4, we will conduct the experiments and analysis defined in the protocol developed in Task 3, and will prepare a final report describing the results. To the extent that the results demonstrate the feasibility of redesign, we will include in this report suggestions for further specification of additional algorithms in a second design phase.

It is hoped that this effort will ultimately result in a commercial-quality high-end hybrid ESBT that

will be able to support the demands of large-scale Ada software development, while conferring the productivity and maintenance benefits that ESBTs for other language environments have previously demonstrated.

2. Approach

2.1 Introduction

In this chapter, the design methodology for the ART/Ada project will be introduced, and the selected design for ART/Ada will be described in detail.

2.2 Design Methodology

2.2.1 Object-Oriented Design

ART/Ada will be designed according to a formal Ada design methodology; this design methodology will be object-oriented design (OOD).

The object-oriented design is an approach to software design in which the system is decomposed into a set of objects. Each object is mapped to one or more Ada packages. Four different kinds of packages will be used in the design:

- Abstract Data Object (ADO)
- Abstract Data Type (ADT)
- Package of subroutines (SUB)
- Package of declarations (DCL)

The Abstract Data Object is a package that contains encapsulated data and operations (expressed as subprograms) performed upon that data. The data is static and local to that package. The data is known as State Data.

The Abstract Data Type is a package that contains an abstract type and operations performed on that abstract type. The operations are expressed as subprograms and the abstract type is declared as the type of one or more parameters of the subprograms within the package.

The package of subroutines is a package of logically related subroutines. There exists no encapsulated data in this package.

The package of declarations is a package of logically related declarations. These declarations may be types, constants, or exceptions.

2.2.2 Reusable Software

Reusable software components will be used in the design of ART/Ada. ART/Ada will capitalize on reusing existing software components provided by the Booch Components [1], a commercially-available library of reusable Ada packages that implement standard data structures and utilities.

2.3 Documentation Techniques

The detailed design will be documented using the specification of Ada packages.

2.4 Impact of Ada

Ada does not support pointer to functions. Therefore, this feature cannot be used for calling functions on the left and right hand side of a rule. Instead, the rule compiler will have to generate a case statement to perform subprogram selection.

Ada does not support conditional compilation. In C, conditional compilation is commonly used to maintain the system-dependent code segments in a single source file. It is also used to add or delete some sections of code depending on user-defined flags. Without the conditional compilation capability, it might be necessary to maintain multiple source files.

Ada provides features such as variant records, generics, tasking, overloading, etc. that are not supported by other programming languages. These features, however, should be used with caution because the costs of using these features may exceed the benefits.

Ada encourages good software engineering practices such as data hiding, object-oriented design, and software reusability.

2.5 Design Approach

ART-IM(the Automated Reasoning Tool for Information Management), formerly called ART/C, is an expert system building tool marketed by Inference. Written in C, ART-IM supports the subset of ART features [2] [3].

ART-IM Version 1.5 will be augmented with an Ada Generator as an Ada deployment option to support the ART/Ada runtime system. As shown in figure 1, the input to the Ada Generator is an ART source file, and its output is an Ada source file. At any point after an ART source file is loaded into ART-IM, the generator can be invoked to generate the Ada source code that will be used to restore ART internal data structures for the ART/Ada runtime system. The ART knowledge base does not have to be reset before the generator is invoked. The ART program can also be run up to a breakpoint before the code generation takes place.

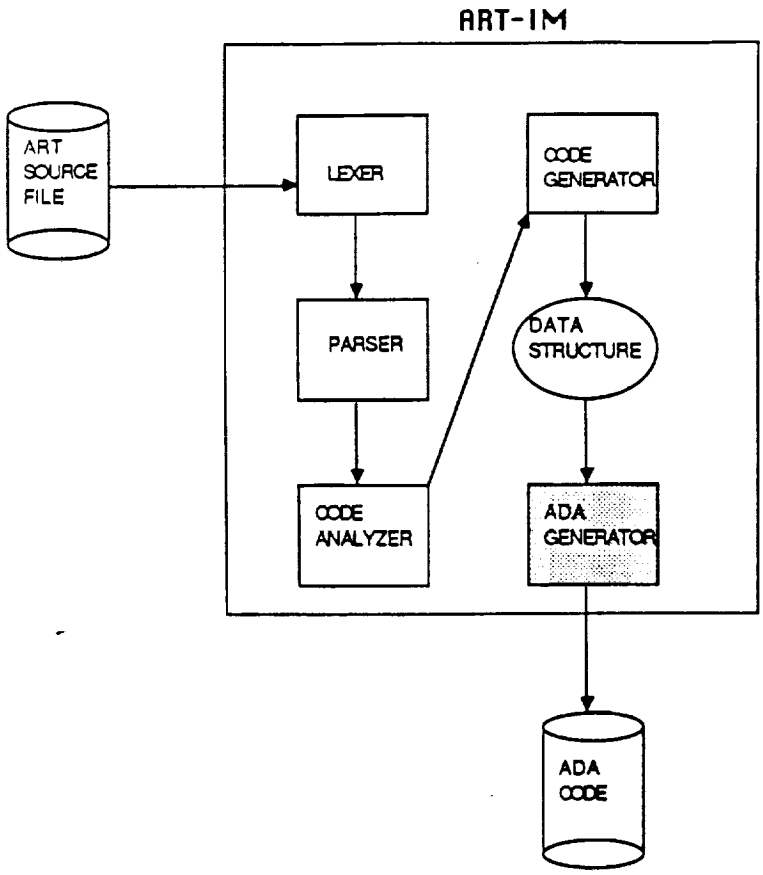


Figure 2-1: Ada Generator

As shown in figure 2, the generator output Ada source code should be compiled and linked with the ART/Ada runtime system and the user-written Ada code to produce an executable image.

This approach offers several benefits:

- It will provide both interpreter and deployment compiler in the same environment. This means that the same environment can be used both for development and deployment.
- It is based on a proven technology.

In order to support the ART/Ada runtime system, it is desirable to add the following additional features to ART-IM 1.5:

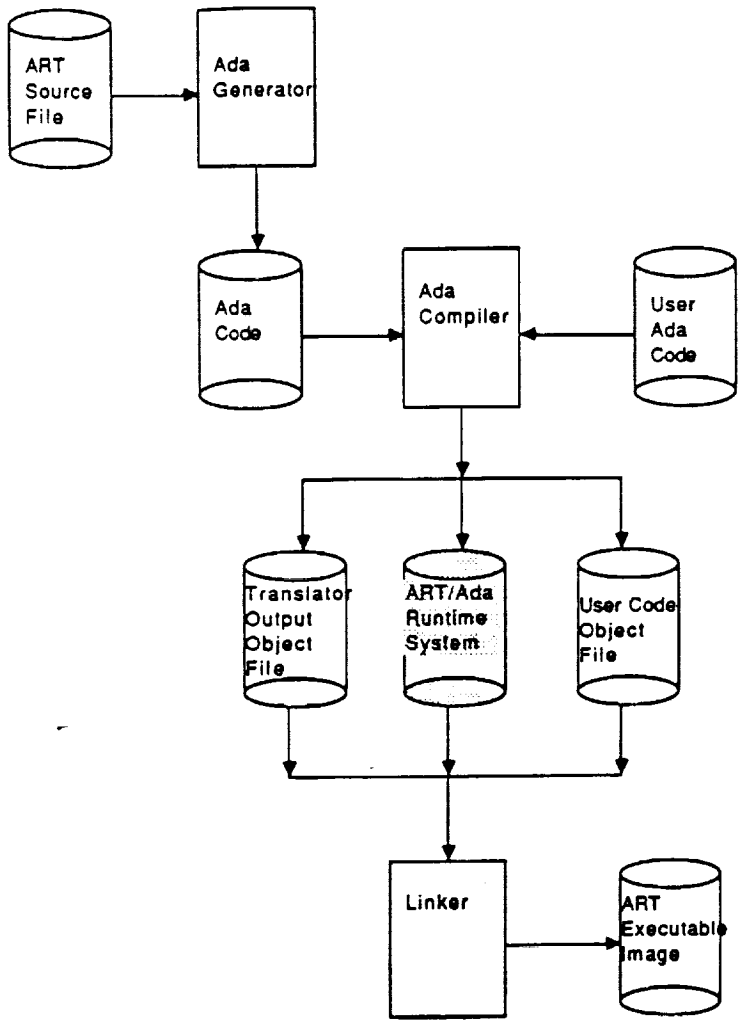


Figure 2-2: ART/Ada Runtime System

- Ada bindings to call in from Ada to ART-IM
- Strong type checking capability

In order to support the runtime type checking, ART-IM should be enhanced with a facility which prevents a fact with a wrong type from entering into the ART database.

In addition to this, the ART-IM parser should be enhanced to enforce type checking at compile time. This is analogous to the type checking of Ada compilers. In order to support this, the syntax of defrelation should be extended to have an additional information on the data type. For example,

```
(defrelation foo ((?x :integer) $(?y :symbol) (?z :float)))
```

Inclusion of these features in the ART/Ada prototype is highly desirable, but it may not be possible because of the time limitation.

A scenario of how one would develop an ART/Ada application is as follows:

1. Develop the knowledge-base on ART-IM
2. Callout to Ada from ART-IM using the standard callout mechanism for both ART-IM and ART/Ada
3. Translate the knowledge-base from ART-IM to Ada using the Ada Generator option in ART-IM
4. Compile the Ada code generated by the Ada generator and link it with the ART/Ada runtime system and the user's Ada code
5. Deploy the Ada executable image on a host computer or a target system

2.6 Differences between ART-IM 1.5 and ART/Ada 1.0 Prototype

The ART/Ada 1.0 prototype will exhibit a high degree of compatibility with ART-IM 1.5, which is the base language for ART/Ada. However, there will be some differences.

The following is a list of features which will be different from ART-IM 1.5:

- Data Type: ART/Ada 1.0 will support Ada double precision float (64 bits) whenever an Ada compiler supports it. The VAX Ada compiler supports it as LONG_FLOAT. The Verdix Ada compiler supports it as FLOAT. The HP Ada compiler does not support it.
- User Interface: ART/Ada 1.0 will have a command loop which supports a subset of the ART-IM user interface. The minimal user interface support is required to debug ART/Ada and its applications. The command loop can be easily removed for an embedded system.
- Input/Output: Input/output capability of ART/Ada 1.0 will be limited to print, princ, and printout. Streams will not be supported.
- Memory Management: ART/Ada 1.0 will use the Ada "new" statement to allocate memory, and use the Ada generic procedure, unchecked_deallocation to deallocate memory.
- Exception Handling: ART/Ada 1.0 will not have the ART-IM-style debugger, but will instead rely on the exception-handling mechanism of the Ada language.
- Math package: ART/Ada 1.0 will not support math functions.
- Justification: ART/Ada 1.0 will not support the justification facility.
- Logical Dependency: ART/Ada 1.0 will not support the logical dependency facility.
- Schema System: ART/Ada 1.0 will not support the schema system.

3. Design Overview

3.1 Ada Generator

3.1.1 Introduction

Instead of developing a new front-end program in Ada to generate the Ada source code from an ART program, the Ada Generator module will be developed in C and added to ART-IM as an option. The Ada generator will take as input the internal data structures of ART-IM that define the knowledge base, and generate as output one or more Ada packages that will be used to initialize ART internal data structures for the ART/Ada runtime system.

The output of the Ada generator will be compiled and linked with the ART/Ada runtime system which will be implemented in Ada.

The addition of the Ada generator will make ART-IM ideal as an ART/Ada application development environment. After the development is done, the same environment can be used to generate Ada code for deployment.

3.1.2 Ada Generator Modules

The Ada generator is composed of the following modules:

- Code Vector Dumper: This module dumps code vectors for function calls in LHS or RHS which will be interpreted by the function call interpreter of the runtime system.
- Pattern Network Dumper: This module dumps the pattern network which will be interpreted by the pattern network interpreter of the runtime system.
- Join Network Dumper: This module dumps the join network and its associated data structures such as instantiations, partial matches, and agenda which will be interpreted by the join network interpreter of the runtime system.
- Miscellaneous Dumper: This module dumps rule information and deffacts information, symbol table, and other miscellaneous information.

3.2 ART/Ada Runtime System

3.2.1 Introduction

The ART/Ada runtime system is modeled after ART-IM runtime system. When it is compiled and linked with the ART/Ada runtime system, the Ada code generated by the Ada Generator will restore the ART internal data structures for the ART/Ada runtime system.

3.2.2 Runtime System Modules

The ART/Ada runtime system is composed of the following modules:

- ART Object Manager: This module contains symbol table and related functions such as symbolp, stringp, numberp, integerp, floatp, factp, sequencep, streamp, type-of, eq, equal, gentemp, string-append, string-to-symbol, symbol-to-string, length$, position$, member$, nth$.
- Database Manager: This module contains ART database management subprograms (e.g. assert, retract, etc.)
- Function Call Interpreter: This module interprets function calls in LHS or RHS using code vectors. It also handles user-defined call-out.
- Pattern Network Interpreter: This module interprets the pattern network.
- Join Network Interpreter: This module interprets the join network.
- Agenda Manager: This module contains Agenda management subprograms.
- Inference Engine: This module contains subprograms that controls the inference engine (e.g. run).
- User Interface: This module provides a minimal subset of the ART-IM command loop. ART/Ada will support watch/unwatch, reset, run, agenda, facts.
- IO package: This module contains ART I/O functions. ART/Ada will support print, princ, and printout. ART/Ada will not support streams.
- Error Handler: This module defines the top level error handler.
- Call-in/Call-out package: This module defines subprograms that handle call-in and call-out.

References

1. Booch, G. *Software Components With Ada.* Benjamin/Cummings Publishing, 1987.

2. Inference Corporation. *ART Version 3.1 Reference Manual.* Inference Corporation, 1987.

3. Inference Corporation. *ART-IM 1.0 Reference Manual.* Inference Corporation, 1988.