N93-11928

An Embedded Rule-Based Diagnostic Expert System in Ada.

Robert E. Jones
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

Eugene M. Liberman
Sverdrup Technology Corporation
2001 Aerospace Parkway
Brook Park, Ohio 44142

ABSTRACT

Ada is becoming an increasingly popular programming language for large Government-funded software projects. Ada with its portability, transportability, and maintainability lends itself well to today's complex programming environment. In addition, expert systems have also assumed a growing role in providing human-like reasoning capability and expertise for computer systems.

This paper discusses the integration of expert system technology with Ada programming language, specifically a rule-based expert system using an ART-Ada (Automated Reasoning Tool for Ada) system shell. The Inference Corporation developed ART-Ada under a program sponsored by the NASA Johnson Space Center. The NASA Lewis Research Center was chosen as a beta test site for ART-Ada. The test was conducted by implementing the existing Autonomous Power EXpert System (APEX), a Lisp-based power expert system, in ART-Ada.

Three components, the rule-based expert system, a graphics user interface, and communications software make up SMART-Ada (Systems fault Management with ART-Ada). The rules were written in the ART-Ada development environment and converted to Ada source code. The graphics interface was developed with the Transportable Application Environment (TAE) Plus, which generates Ada source code to control graphics images. SMART-Ada communicates with a remote host to obtain either simulated or real data. The Ada source code generated with ART-Ada, TAE Plus, and communications code was incorporated into an Ada expert system that reads the data from a power distribution test bed, applies the rules to determine a fault, if one exists, and graphically displays it on the screen.

The main objective of this study, to conduct a beta test on the ART-Ada rule-based expert system shell, was achieved. The system is operational. New Ada tools will assist in future successful projects. ART-Ada is one such tool and is a viable alternative to the straight Ada code when an application requires a rule-based or knowledge-based approach.

Keywords: Ada, Expert System, Rule-Based, Knowledge-Based.

INTRODUCTION

Ada is becoming the language of choice for large Government-funded projects as increasing software size and complexity drives the cost of development and maintenance upward. Ada (J.G.P Barnes, 1984) language offers standardization, portability, maintainability, and readability. All these features provide an excellent environment for developing complex software, increasing programmers' productivity, and encouraging team work.

Many large-scale software projects targeted toward space applications involve health monitoring and control of mission-critical systems. The necessary expertise to maintain reliable space mission operations is often unavailable due to limited resources. Even if the expertise is available, routine system health monitoring remains a time-consuming and tedious task. Expert systems have been taking an increasingly larger role in providing knowledge for problem resolution and in handling tedious tasks.

The integration of expert system technology with Ada programming language results in a powerful combination for use in many large-scale computer applications. An expert system shell that generates Ada code is one such combination. ART-Ada is an expert system shell that generates Ada code.

This paper describes the techniques for and implementation of a rule-based expert system using the ART-Ada expert system shell. The Inference Corporation developed ART-Ada under a program sponsored by the NASA Johnson Space Center. The NASA Lewis Research Center was chosen as a beta test site for ART-Ada. The test was conducted by implementing the existing Autonomous Power EXpert system (APEX) (Ringer and Quinn, 1990) with ART-Ada. APEX is a Lisp-based power expert system designed as a fault diagnostic adviser for the monitoring and control of 20 kHz power distribution test bed. APEX is being developed at the NASA Lewis Research Center.

SYSTEM DESCRIPTION

The system description is limited to a discussion of the software implementation of SMART-Ada and the hardware on which the software was implemented. SMART-Ada is written on a SUN SPARCstation 1 equipped with 16 megabytes of memory and approximately 0.5 gigabyte of hard disk capacity.

The operating system is SUN OS 4.0.3c running X Window X11R3. The following software packages were used to implement SMART-Ada:

(1) ART-Ada
(2) Transportable Application Environment (TAE) Plus
(3) Remote procedure call protocol (RPC)
(4) X Window system
(5) X Window Ada bindings
(6) VERDIX Ada compiler

ART-Ada is an expert system shell for the development of rule-based or knowledge-based expert systems. ART-Ada is based on the Automated Reasoning Tool for Information Management (ART-IM). One important feature of ART-Ada is its ability to generate Ada source code from a knowledge-based or rule-based application written in ART-IM. The syntax of ART-Ada is compatible with ART-IM, making code developed in ART-IM transportable to ART-Ada as long as no machine-dependent features of either software are used.

TAE Plus is a software package developed by the NASA Goddard Space Flight Center. The package is an integrated environment for creating and running window-based applications with a graphical point-and-click user interface. TAE Plus is based on the X Window System. TAE Plus was chosen for the SMART-Ada graphical user interface because it can generate Ada code.

RPC is a communications protocol developed by Sun Microsystems. This protocol allows machines of different types to interact with each other on a procedure level. This interaction means that one machine can call a procedure on the other, pass arguments to the procedure and then receive any returned values. RPC software for SPARCstation 1 was developed in the C programming language. An Ada-to-C interface is required to allow SMART-Ada software to take full advantage of RPC protocol. This interface was written for this project. The X Window System, developed at MIT, has become the industry standard and runs on a wide range of computing and graphics machines. The X Window System provides a powerful windowing environment for producing high-performance graphical interface. Since the X Window System was developed in C programming language, an interface between the Ada code and the X Window C code is necessary to take advantage of the X Window System features.

Ada language bindings to the X Window System are a package developed by Science Applications International Corporation (SAIC). The bindings provide the necessary interface between X Window C libraries and the Ada code in SMART-Ada. The use of bindings allows the Ada code to take full advantage of the powerful X Window System procedures.

The VERDIX compiler is used for compilation and executable code generation. The VERDIX compiler was recommended by the Inference Corporation for development and implementation of SMART-Ada. The VERDIX compiler also allows for a good Ada-to-C interface, which is an important feature needed for accessing X Window System procedures with the Ada code.

SMART-Ada system components

The SMART-Ada system consists of three major components:

(1) A rule-based expert system
(2) A graphics user interface (GUI)
(3) Communications software

These three components were integrated to make up the entire SMART-Ada system. All components are implemented in Ada programming language. However, in two cases the interface between C libraries and the Ada code is used to enable Ada to access existing C software libraries. The interface between X Window procedures and the Ada graphics programs was implemented with the SAIC Ada language bindings to the X Window System.

The interface between RPC software and Ada was a part of SMART-Ada development. Each component and its implementation is discussed in greater detail in the following paragraphs.

Rule-Based expert system

The knowledge-based rule set was developed with ART-IM because of its user-friendly development environment. ART-IM's powerful debugging features were heavily utilized.

The rule-based expert system is responsible for monitoring the operational state of a 20 kHz power distribution test bed. If an abnormality occurs in the test bed, the expert system detects the fault condition and isolates the probable cause. It performs fault detection by comparing measured operating values to the expected values, accessing information and rules contained within its knowledge base in order to isolate the fault.

A collection of rules in the knowledge base forms groups of logical subtasks. The logical subtasks are connectivity, initialization, detection, isolation, affected loads, and recommended actions. The connectivity subtask determines the power distribution configuration of the 20 kHz power distribution test bed. The initialization subtask calculates expected values for voltages, currents, and power on the basis of the test bed configuration and the physical properties of the test bed components. The detection subtask compares the expected values obtained in the initialization subtask with the actual system values obtained from the test bed. The isolation subtask isolates the problem and assists the expert system in determining where in the circuit problems have occurred. The affected loads subtask determines which load is affected by the faults in the system. The recommended action subtask, in the future development of the system, will recommend a new configuration for the power distribution or will automatically reconfigure the power distribution to alleviate existing problems.

The order of rule execution in a subtask is determined by the salience of each rule. The higher salience rules execute first and the lower salience rules execute last. The last rule in the subtask contains the lowest salience so that it will execute only after all the rules in the subtask have executed. In SMART-Ada, the lowest salience rule in each subtask is used to "clean up" the current subtask and set the environment for the next subtask. Since SMART-Ada is a monitoring system, the rule-based expert system must execute continuously. The last subtask sets the environment for the first subtask, creating the effect of an infinite loop.

Expert system and Ada Interface

In order to execute Ada language subprograms from ART-Ada, an interface between ART-Ada and Ada language is required. The interface is accomplished by defining an Ada USER package (Figure 1). Within ART-Ada, USER is a reserved symbol for accessing external Ada code from ART-Ada rules. Parameter passing is also possible between ART-Ada and subprograms in the USER package. The parameters, however, have to adhere to the syntax of ART-Ada. SMART-Ada uses subprograms contained in the USER package to control the graphical user interface and to obtain data from the 20 kHz power distribution test bed.

ART-Ada contains a feature that allows one function to be specified as asynchronous. A function defined as asynchronous is automatically invoked before and after each rule execution. The subprogram named ASYNCH_FUN in the USER package is defined as an asynchronous function. ASYNCH_FUN is responsible for reading data from the test bed or using simulated data and providing data to the rule-based expert system (Figure 2) as well as to the graphics interface. All subprograms in the USER package are capable of accessing and modifying data in the ART-Ada code.

As the rules are being executed, the asynchronous function waits until the last rule has been completed. The last rule execution is an indicator for ASYNCH_FUN to read new data from the 20 kHz power distribution test bed and introduce the new data to the expert system for evaluation. The new data are also dispatched to the graphics interface for a system status update. The execution of the expert system then continues with execution of the first subtask.

The WRITE_TO_FILE subprogram in the USER package is responsible for writing the explanations of errors that occur as a result of the rule-based expert system execution. The rule-based expert system provides the file name and the line-by-line text that is to be entered in the file specified in the file name parameter. The file name is determined by which rule subtask is executing at the moment. The detection rule subtask, for example, writes to the DETECT.DAT file, and the isolation rule subtask writes to the ISOLATE.DAT file. The text that goes in a file is determined by the executing rule. The FLAG_ERROR subprogram is invoked when an error condition is present. The rule that invokes the subprogram must provide the switch name and the faulty component to the subprogram. The code in the FLAG_ERROR subprogram communicates by means of the graphics interface and sets the visual alert in the graphics module.

Graphics User Interface

The graphics user interface (GUI) for the SMART-Ada was developed with TAE Plus. TAE Plus was chosen for this application because of its capability to generate Ada code for the developed graphics.

The DDO (data driven object) feature of TAE Plus is used to achieve the dynamic display of the system status. The screens and the DDO's were created with the TAE Plus graphics editor. The screens show the system status at high and low levels. The high- level screen shows the overall system state and configuration. The faults with the system components are shown with appropriate color and a warning banner. The lower-level screens provide a more detailed look of a selected component. The currents, voltages, and power indicators contain the up-to-date values. The faults with any of the values are also marked with appropriate colors and a warning banner.

An explanation for the errors is also provided. The Rule-Based expert system generates the text corresponding to the system status. The text is stored in the file set up for that purpose. The file DETECT.DAT contains the result of the detection subtask execution, the file ISOLATE.DAT contains the results of the isolation subtask execution, etc. The GUI displays the contents of these files to provide the user with the explanation of the system status.

Communications Software

SMART-Ada communicates with a remote host to obtain either simulated or real data. The communications link between SMART-Ada and the data acquisition software on the remote host is accomplished by using the transmission control protocol / internet protocol (TCP/IP) and RPC protocol. The remote host must contain the procedure that is invoked to provide data to SMART-Ada. The remote procedure returns system status data that are interpreted by the system.

BETA TEST RESULTS

The main objective of SMART-Ada implementation was to conduct a beta test on the ART-Ada rule-based expert system shell. The beta test revealed that it is, indeed, possible to implement an Ada-based application by using the ART-Ada expert system shell. However, a few problems were found as the project developed. The first problem occurred in an attempt to deploy a set of rules. The set of rules was executing properly in the development environment, but when the rules were converted to Ada source code and an executable code was generated, the program failed to run. A constraint Ada error was raised when execution was attempted. The problem was reported to the Inference Corporation who acknowledged and fixed the problem and sent us corrected version of ART-Ada. The second problem was found with the ART-Ada package. A variable contained no value after a value had been assigned to it. The Inference Corporation has acknowledged the problem and promised to fix the package for the next version of the software and has suggested a way to work around the problem.

The SMART-Ada system which was developed consisted of approximately 70 rules in ART-Ada. The code for manipulation of the switches generated approximately 340 Ada statements. The interface between ART-Ada and Ada consisted of approximately 200 Ada statements, while the Ada Graphics interface produced about 2100 statements. The completely deployed system including system overhead produced 13,484 Ada statements total. Further optimization of the ART-Ada code could reduce this number slightly.

The maintainability issues of the ART-Ada code must be addressed. Unfortunately, the tendency is to address maintainability of the Ada code generated from the ART-Ada application. The proper way to maintain the ART-Ada application is through the ART-Ada code itself. System maintenance will become much easier if emphasis is placed on ART-Ada code maintenance rather than on Ada code maintenance.

CONCLUDING REMARKS

As Ada programming language becomes more and more prevalent, new Ada tools will assist in making Ada projects even more successful. ART-Ada is the first rule-based tool available for Ada programming language. The ART-Ada package has its problems, like any initial version of a major software release. But the authors feel that ART-Ada is a viable alternative to the straight Ada code for an application that requires a rule-based or knowledge-based approach.

ACKNOWLEDGMENTS

The authors would like to thank Todd Quinn of Sverdrup Technology Corporation for his invaluable technical and editorial assistance and the entire Space Electronics Division AI laboratory staff for their comments and support.

REFERENCES

1. Barnes, J.G.P (1984). Programming in Ada. 2nd ed., Reading, MA, Addison-Wesley.

2. Ringer, M.J. and Quinn, T.M. (1990), Autonomous Power Expert System. NASA CR-185263.

```
with ART;
package USER is

        function ASYNCH_FUN;

        procedureWRITE_TO_FILE
        (NAME_OF_FILE :in ART.ART_OBJECT;
                STRING_VALUE:in ART.ART_OBJECT);

        procedure FLAG_ERROR
                (SWITCH_NAME:in ART.ART_OBJECT;
                 COMPONENT:in ART.ART_OBJECT);

        end USER;
```

Figure 1. Ada USER Package

```
with USER; use USER;
(def-user-fun asynch-fun
        :args ()
        :returns                :void
        :compiler               :verdix)

(def-user-fun write-to-file
        :args (
        (FILE-NAME     :ART-OBJECT)
        (STRING_VALUE           :ART-OBJECT))
        :returns                :void
        :compiler               :verdix)

(def-user-fun flag-error
        :args (
        (SWITCH         :ART-OBJECT)
        (COMPONENT :ART-OBJECT))
        :returns                :void
        :compiler               :verdix)

        .
        .
        .
        (set-asynch-func async-func)
```

Figure 2. The rule-based expert system use of USER package