Implementing Advanced Artificial Intelligence Concepts in Ada:

A Case Study of a Prototype Expert System for a Real-Time Electronic Warfare Application

Joan Hardy, U.S. Army CECOM Center for EW/RSTA,
Joseph W. Croghan and Myron L. Cramer, Booz, Allen & Hamilton Inc.

1.0 Introduction

The application described in this paper arose from an effort to develop an Expert Power Management System (EPMS) for use with an advanced concept radar jamming system. This program required the merging of expert system technologies, Ada® software design, advanced Electronic Warfare (EW) concepts, and real-time software design concepts. This paper reports how these different areas have been merged within the constraints of an advanced concept demonstration program.

A *power management system* controls the distribution of electromagnetic transmissions from an electronic jamming system used to defeat threat radar systems. Against multiple threat radars, the power management system must identify and prioritize threat emitters, decide which emitters to jam, and must select effective jamming techniques for designated targets. Available jamming power must be shared among assigned jamming techniques considering frequency, timing, and technique waveforms.

1.1 Project Goals

The project was structured to address several areas, corresponding to project goals. These were to: (1) validate the use of an expert system for use as a power management system, (2) include provisions for parallel processing implementations, (3) develop EPMS software in Ada, (4) provide the Army with a usable demonstration, and (5) provide a flexible design architecture that can be adapted to many different subsequent implementations.

1.2 Development Effort

The EPMS development employed an Expert System Shell, CLIPS. This shell was a key element in our approach in that it provided a validated off-the-shelf artificial intelligence product, designed for easy programming language interface. The CLIPS Ada implementation combined with the employment of Ada for the other portions of the software results in a development that retains exceptional transportability. Since EPMS is an advanced concept demonstration program and can impact other advanced concept programs, the ultimate hardware implementations is yet to be determined. Accordingly, transportability of software development is essential.

The development project consisted of several tasks: (1) determination and documentation of functional requirements, (2) design and development of knowledge base, (3) design and development of demonstration system software, and (4) testing of concept.

Washington Ada Conference 1990

2.0 Knowledge Base System Requirements

In order to effectively demonstrate the Expert Power Management System, a support environment, consisting of an environment simulation, a simulation of threat systems, a user interface and a flight path simulation was developed. One design goal was to structure the system so that the majority of the technical effort could be spent on the Expert Power Management System. This section outlines that design and the design tools used.

2.1 Top Level Demonstration System Design

EPMS is envisioned as an executive control unit to provide a higher level of control for the advanced radar jamming system based upon a higher level of situational awareness resulting from (1) integration with other Aircraft Survivability Equipment (ASE), (2) interfaces with aircraft avionics, (3) advanced decision logic to optimize resource utilization against multiple threats.

As noted above, the demonstration system contains five subsystems: the Expert Power Management System, the User Interface, the Environment Simulation, the Adversary Doctrine, and the Flight Path Simulation. In addition to these software subsystems there is a set of databases that are used by the software. The relationships of these software modules and databases are shown in figure 1, along with the data flows between the modules.

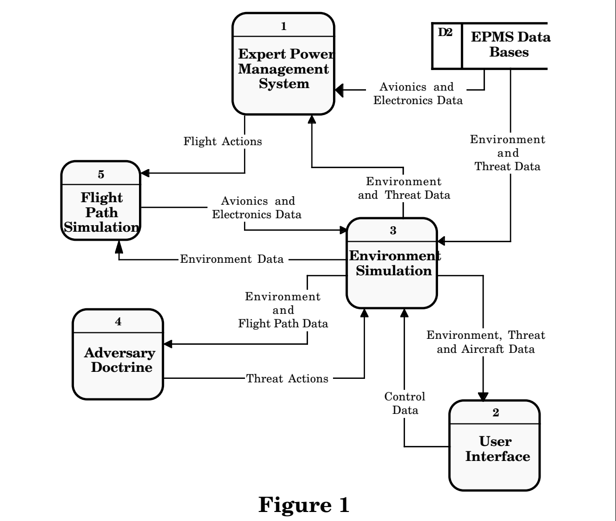
1
Expert Power Management System
D2
EPMS Data Bases
Avionics and Electronics Data
Flight Actions
Environment and Threat Data
5
Flight Path Simulation
Avionics and Electronics Data
Environment and Threat Data
3
Environment Simulation
Environment Data
Environment and Flight Path Data
4
Adversary Doctrine
Threat Actions
Environment, Threat and Aircraft Data
Control Data
2
User Interface

Figure 1

Each of these components plays an important role in the system operation. The *Environment Simulation* maintains data about events and objects external to the aircraft; this includes primarily threat radars and systems. The *Adversary Doctrine* module uses data from this simulation to control the actions of the threat radars and and systems. The *Flight Path Simulation* maintains data on aircraft-related events and objects, including avionics data, the receiver data and the jammer information. The *Expert Power Management System* uses data from the Flight Path Simulation, as well as environment data to control the actions of the aircraft's jammers and receivers. Finally, the User Interface displays the critical information from the other system components as well as providing a mechanism for user input and control of the system.

Since the primary purpose of this effort is to develop an Expert Power Management System, the development of this module was undertaken first. The following section describes the issues involved in the Expert Power Management System requirements analysis and design.

2.2 Expert System Requirements

The requirement for the EPMS is to specify expert knowledge about the effective use of radar jamming equipment as a knowledge base, such that the knowledge may be used to automatically manage the use of that equipment. The advantage of an expert system is its ability to apply diverse, largely unstructured knowledge to a problem. The problem to be addressed by the EPMS is made up of such knowledge. The EPMS must balance mission, flight context (friendly and opposition forces and actual location, terrain, etc.), equipment condition and availability, and the actual threats that must be countered.

The major aspect of the problem is resource allocation. A number of threats must be countered as effectively as possible with a limited amount of equipment and power. This type of problem is amenable to a number of solutions, including linear programming as well as expert systems. Complicating this issue, however, are a host of constraints that must be taken into consideration. Traveling wave tubes must not be allowed to overheat. Jamming techniques must not be allowed to inadvertently draw friendly units under fire. Threats must be addressed according to a set of priorities. Transmitters can only operate over certain bandwidths, and must be applied to threats in an appropriate manner. These issues must be addressed in concert with the basic resource allocation problem, in a coherent and efficient manner.

The total problem is sufficiently complex to require non-conventional algorithms. Moreover, it is of a form that lends itself to an expert systems approach. The implementation will represent the knowledge of experts in the field of electronic warfare, providing an electronic expert, EPMS, capable of reacting swifter than any human expert.

Many expert systems are built using expert system shells. These are software tools designed to support the construction and/or execution of expert system software. In some systems, the shell is the equivalent of a language interpreter and the entire expert system is written in the shell "language." In other cases the shell represents only a portion of the finished system. This may take the form of an embedded shell kernel within the system, or the shell may generate source code for the expert system that will be compiled by the same compiler used for the finished product.

Typically expert system shells provide a high-level language to represent "rules". Similar in form to a conventional programming language *if..then* construct, a rule specifies a set of conditions to which it applies and a set of actions to be accomplished when the conditions are met. Unlike its conventional cousin, a rule is conceptually a parallel construct, evaluated along with other rules against current context and not a part of a sequential process or decision tree.

Rules lend themselves to processes involving unrelated, specific actions to be taken in particular situations. The EPMS problem presents a series of special cases and techniques that must be used only when certain conditions are true. The conditions, in this problem, are elements such as the location of friendly and opposition units, the status of onboard equipment, the current threat environment and the current mission and mission status. The rules represent the relationship between a perceived opposition unit and its threat, the relationship between the operation of equipment and its future utility, the probability of particular techniques being effective against threats, and the requirements of the mission.

In some cases, the use of rules is not sufficient to a particular problem, and other techniques must be used. Depending upon the shell, this may require representation of the technique using rules, extension of shell capability, or special software developed for the particular technique. One example of such a technique would be the use of constraint propagation networks to represent the relationship between power demand, available power, and the degeneration of equipment such as the traveling wave tubes (TWT) due to overuse. Another way to approach the overheating of TWTs might be the use of temporal logic to specify the relationship between equipment usage and equipment availability. The first technique would require special code to be developed in the expert system shell while the second technique would lend itself to rule specification.

Hardware hosting EPMS will change over time, as a result of its advanced concept stage. Three versions of the EPMS processors are envisioned at this stage: the development system, an operational system, and an alternative version of the operational system using a parallel architecture.

The development hardware was a conventional microcomputer, an 80386 MS-DOS microcomputer. By using a conventional machine, the full range of tools and utilities available under MS-DOS were used to increase productivity. Additionally, development on an MS-DOS compatible machine has facilitated hosting the demonstration model on similar machines.

The actual configuration of the operational hardware is open. The assumption at this point is that a general-purpose microcomputer will be used. The basic requirement is that it be a target for some Ada cross-compiler, so that the EPMS code may be ported to it at the appropriate time.

Apart from selecting the actual processor used for the EPMS, interface with EW system hardware will be required. The EPMS software will be carefully modularized to avoid contaminating general-purpose code with implementation-specific issues. As a general rule, it is assumed that communication between the EPMS and other EW and avionic systems will be via data buses. The details of these bus designs are left for future development.

The major requirement for any real-time embedded system is performance. One method of providing a high-performance system is to employ multiple processors to share the processing load.

The use of any parallel processing architecture affects the structure of software. These were considered in developing the EPMS functional design and in grouping and apportioning processes.

2.3 Provisions for Parallel Processing

In developing the design for EPMS, provisions were included for future parallel processing implementations. These provisions began with the initial partitioning of system functional requirements. The resulting breakout of the EPMS design included separate functional elements for (1) the Expert System, (2) the Input/Output (I/O) Processor, (3) Receiver Manager, (4) Jammer Manager, and (5) the Tracker. The objective of these partitions of the processing work load is to unburden the Expert System as much as possible from routine activities that might otherwise distract it from situation analysis and decision making responsibilities in a real-time environment.

The I/O Processor will handle incoming and outgoing communications between EPMS and the other EW and avionics systems. This element attends to the communication requirements of the on-board data busses, forwarding commands and receiving data reports.

The Receiver Manager supervises the use of the available receiver assets, issues measurement commands as directed to by the Expert System and handles status reporting from these receivers.

The Jammer Manager controls the use of the available jammers, issues jamming assignments as directed by the Expert System.

The Tracker conducts mathematical calculations in support of jamming activities as assigned by the Expert System.

2.4 Other Software Components

The other four software modules in the demonstration system were designed in such a way that is compatible with the Expert Power Management System, however since these other components constitute a support environment in which to operate the Power Management System, they are not intended for use in the objective EPMS system. This means that some of the constraints about development languages or run-time speed are less important. After careful examination, it was determined that developing all software components in the same language would be beneficial. This would allow increased portability and less development overhead. Since it is not usually difficult to integrate expert system shells with the language the shell is written in, it was decided that it would be acceptable to create some modules in Ada and others in a shell written in Ada, such as CLIPS.

3.0 Software Development

In order to effectively develop software for the Expert Power Management System and the support environment, an appropriate development environment was chosen. This section outlines the development environment options and the configuration chosen for this application.

3.1 Software Design Tools and Methodology

A challenging part of the EPMS development effort was selecting a software design that met the constraints of the software development resources and satisfied the technical requirements of the EPMS.

It was an objective to have a design tool that could be easily used by all of the people involved, software developer and subject matter experts, so that the design could be easily revised. *Natural Architect Workstation* by Software AG was selected. This packages allows the rapid development of data flow and control flow diagrams, as well as building a data dictionary from the diagrams. Figure 1 of this paper was created using Natural Architect Workstation. With this tool, the design was easily revised and the results of design meetings quickly documented. In fact, some design meetings actually took place in front of the computer and the changes were made immediately.

3.2 Development Environment Options

Traditionally, expert systems have been constructed using flexible languages embedded in high-productivity development environments. Special-purpose hardware and software supported rapid prototyping and high quality graphics-based user interfaces. More recently, as expert system technology has advanced and applied more and more to real-world issues, it has been necessary to move from these special-purpose development environments to more generic ones.

EPMS is a prototype of a real-time embedded system. In the late 1970's an effort was initiated to develop a language for just this type of system, resulting in the Ada programming language. Recognizing the need for language elements dedicated to embedded systems, the language designers provided mechanisms for multi-tasking (either time-sliced or true parallelism), interrupt-driven code, and direct hardware access.

The driving factor behind the development of the language, however, was the proliferation of languages and development environments. To this end, Ada compilers must be validated against strict criteria before they are allowed to bear the name "Ada," which has been trademarked by the Department of Defense. The benefit to this project is that the language is standard, providing a stable base on which to build the EPMS. Since the final hardware configuration is still open, the choice of a standard language is of utmost importance.

The implementation of an expert system in Ada might be accomplished on a number of levels: using an expert system shell written in Ada, using a set of library functions in Ada, generating Ada code from expert system constructs, or hard-coding expert system algorithms in Ada by hand.

Using an expert system shell written in Ada provides the benefits of the expert system shell while providing the portability and other benefits of Ada. The implementation of the expert system shell may be done consistent with accepted Ada programming style. One potential drawback is that the shell may not provide all of the necessary functionality or performance for a particular project, requiring changes or extensions to the shell. Thus source code, or some usable mechanism for extending the shell, is desirable. One such shell, CLIPS, is described in more detail in Section 3.2.

Using a set of library functions in Ada provide the benefit of rapid development, while maintaining the speed and flexibility of Ada. A typical place where this is employed is the Graphical Kernel System (GKS), used for user interface building. By using this system, a user interface can be rapidly generated in Ada. GKS is described in more detail in Section 3.3.

Generating Ada code from expert system constructs is a compromise between the first positions. These constructs (rules, constraint networks, etc) would be specified as if for a shell. In some cases, a shell would be used to test the constructs interactively. When necessary, the constructs would be translated into Ada, allowing them to be compiled and executed at the speed of compiled programs. This approach provides some of the advantages of the shell approach and eliminates some of the disadvantages. Unfortunately, many of the disadvantages of the translation would also be introduced.

Hard-coding expert system algorithms in Ada by hand may well be the method most in the spirit of the language. As support for reusable code modules provides libraries of expert system algorithms and data structures, this option will become more prevalent. However, at the current time the expense associated with building an expert system from scratch would preclude any type of prototype and iterate development methodology.

Requirements dictate the fastest system, but contractual constraints of an advanced concept program prohibited spending too much time developing custom tools for this project. Moreover, the choice of tools already developed and available is very small, due to inertia on the part of the artificial intelligence community with respect to Ada. In order to develop an effective expert system, some combination of the above approaches may be required in similar programs. This will allow other developers to capitalize on the advantages of some of the approaches in situations where the disadvantages are not significant.

3.3 C Language Integrated Production System (CLIPS)

The Artificial Intelligence Section (AIS) at NASA/Johnson Space Center (JSC) has encountered a number of problems delivering LISP-based expert systems to NASA users. Three problems were especially detrimental to the use of expert systems within NASA: low availability of LISP on most computers, high cost of specialized LISP tools and hardware, and poor integration of LISP with other languages. These factors all made embedded applications difficult.

To solve these problems, the NASA/AIS developed an expert system tool written in and fully integrated with the C language: the *"C" Language Production System* (CLIPS). CLIPS was designed specifically to provide high portability, low cost, and easy integration with external systems. CLIPS is also available in the Ada language, and as with C the Ada version is easy to integrate with external Ada programs. The primary representation methodology in CLIPS is a forward chaining rule language based on the Rete Algorithm. CLIPS provides reasonable performance on a wide variety of computers. It also includes tools for debugging. CLIPS in in use a most NASA centers, numerous universities, and many places in private industry. It is available at a low cost through NASA.

Ada-CLIPS is an expert system shell descended originally from Inference Corporation's *ART*™. The Ada version of CLIPS was converted from the C version of CLIPS, which was based loosely on the concepts embodied in ART. CLIPS provides a fairly rich expert system development environment, primarily based around a forward-chaining inference mechanism which uses the Rete Algorithm. This is a very traditional expert system environment. Source code is provided with the system and instructions are provided for extending it to handle additional functionality.

3.4 CLIPS Development Environment

In applying CLIPS, an application developer encodes domain rules in CLIPS syntax. This syntax is fairly easy to learn and provides immediate results that can be evaluated for accuracy. While the encoding of the rules is simple, the flow of control in a rule based system is usually difficult to predict, since CLIPS decides the best order in which to fire the rules. Some time can easily be spent attempting to determine why a particular rule is not firing at the expected time, or why it is firing when it was not expected. CLIPS provides some debugging tools to help work these issues.

On the PC the Ada version of CLIPS offers a conventional development environment. CLIPS commands are interpreted from a command-line interface. For example: to see facts, the command "*(facts)*" is entered; to view the rule agenda, the command "*(rule agenda)*" is entered. Other development features are invoked similarly. CLIPS allows for loading predefined files, facilitating development. The "C" language version of CLIPS provides additional development support for both the PC and the Macintosh implementations. Both of these versions support a window system that can show the facts list, the agenda and rules as changes occur during a run. In order to utilize these enhanced environment features, we tested the method of generating CLIPS code using one of the "C" versions and then running this code under the Ada version of CLIPS. We found that this method worked well, although there are some minor incompatibilities among the three versions.

3.5 CLIPS Execution Environment

The CLIPS rule bases can be either executed directly using the CLIPS interpreter or called from an Ada function. In the case of this project, CLIPS is called from an Ada function, namely, the User Interface. The executable for the software is an Ada executable, this function sets up the user interface and calls CLIPS. CLIPS then runs, firing all appropriate rules, and makes periodic calls back to the Ada function in order to update the user interface. When the user gives input to the interface, that data is passed to CLIPS. There is no operational difference between this method of operation and starting the CLIPS interpreter, loading the appropriate files and running the rules base from there. In the application described in this paper, Ada was chosen to control the processing since User Interface is the top level function for the demonstration system.

3.6 Graphics Kernel System (GKS)

The main purpose of the Graphical Kernel System is the production and manipulation of graphics information. These graphics can vary from simple line graphics to images in grey scale or color. GKS allows programs to support a wide variety of graphics devices and is defined independently of programming languages and was adopted as an International Standard by ISO in 1985 for computer graphics. By using a library of grahics functions and a set of bindings appropriate for the development language, a programmer can use GKS from many different languages. For the EPMS demonstration, the Ada implementation was used.

GKS allows a programmer to develop a graphical interface much more rapidly than could be done from scratch. The advantage of using GKS over another system is its portability and maintainability resulting from the strict standards of GKS.

3.7 Completing the Development Environment

Selecting a development environment involved two primary decision criteria: the flexibility to model the necessary constructs; and ease of software development and modification. Many times these two criteria are mutually exclusive. As a particular development environment provides an increasing set of predefined functions and features it facilitates the development and modification of software, however, it may also become difficult to represent very specific constructs since the environment is less flexible. It is particularly important to be able to develop and modify software quickly to facilitate the development of the EPMS prototype. Rapid iteration cycles with the domain experts are critical to system success.

A secondary decision criteria, system speed is not as important as those criteria listed above since many parts of the demonstration system are not intended for integration in the objective EPMS. However, for the prototype to operate effectively and be useful, it must run in a reasonably quick manner.

Given the above decision criteria it was decided that the system would be built entirely in CLIPS with the exception of the User Interface which would be written in Ada using the GKS system. This allows for rapid prototyping of the User Interface, the Support Modules and the Expert Power Management System. CLIPS facilitated the rapid prototyping and evaluation of the domain knowledge base and decision rules. Comparable rapid prototyping would have been more difficult in Ada. CLIPS carries some overhead associated with it and is not as fast as a system coded directly in Ada might have been, however in this case the development issues easily outweighed the performance issues.

4.0 Conclusions

Based upon our experience with this development effort, we have drawn the following conclusions:

(1) The use of an expert shell can offer benefits for advanced concept programs. The use of CLIPS in this program allowed engineering efforts to be concentrated in the application-specific areas, rather than in extensive software development. These engineering efforts were concentrated on the determination of system functional requirements and on analysis and decision rules.

(2) The use of a language standard such as Ada allowed for a high degree of machine independence. In an advanced concept program such as the one described herein, the objective system is yet to be determined. By demonstrating advanced processing concepts in a portable language, there is increased likelihood that software developed will be easily adapted to follow-on advanced concept programs and ultimately into an objective system.

(3) The use of Ada Graphic Kernel System (GKS) allowed for the incorporation of color graphic presentations that enhanced the presentation of demonstrations through a standard graphics system, although the features offered were somewhat limiting.

(4) Overall software development costs were small, considering the scope and innovation required in this project.