2

Proceedings of the Association for Computing Machinery Special Interest Group for Ada Artificial Intelligence Working Group 1992 Summer Workshop

M 93B0000072
June 1993

Janet Faye Johns

MITRE
Bedford, Massachusetts

June 1993

Proceedings of the Association for Computing Machinery
Special Interest Group for Ada Artificial Intelligence
Working Group 1992 Summer Workshop

Janet Faye Johns

The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420

M 93B0000072

same as above

same as above

On 24-27 June 1992, the Association for Computing Machinery (ACM) Special Interest Group for Ada (SIGAda) Artificial Intelligence Working Group (AIWG) held a workshop to discuss Ada real-time and Artificial Intelligence (AI) issues. Workshop discussions covered blackboard architectures, experiences and lessons learned, real-time development approaches and issues, and Ada 9X issues for AI systems. These proceedings included papers by workshop participants describing their large-scale AI with Ada systems. A summary of the related workshop experiences and lessons learned discussions in the areas of requirements analysis, design methodologies, development techniques, test and validation, and maintainability are included.

Ada systems, Artificial Intelligence

Unclassified	Unclassified	Unclassified	Unlimited

Proceedings of the Association for Computing Machinery Special Interest Group for Ada Artificial Intelligence Working Group 1992 Summer Workshop

M 93B0000072
June 1993

Janet Faye Johns

Contract Sponsor N/A
Contract No. N/A
Project No. G30D
Dept. G033

MITRE

Bedford, Massachusetts

ABSTRACT

On 24-27 June 1992, the Association for Computing Machinery (ACM) Special Interest Group for Ada (SIGAda) Artificial Intelligence Working Group (AIWG) held a workshop to discuss Ada real-time and Artificial Intelligence (AI) issues. Workshop discussions covered blackboard architectures, experiences and lessons learned, real-time development approaches and issues, and Ada 9X issues for AI systems. These proceedings include papers by workshop participants describing their large-scale AI with Ada systems. A summary of the related workshop experiences and lessons learned discussions in the areas of requirements analysis, design methodologies, development techniques, test and validation, and maintainability are included.

EXECUTIVE SUMMARY

Artificial Intelligence (AI) with Ada is a reality! Based on the information presented at the 1992 Summer Association for Computing Machinery (ACM) Special Interest Group for Ada (SIGAda) Artificial Intelligence Working Group (AIWG) workshop, Ada is a viable language for AI applications. This workshop provided a valuable opportunity to accumulate more empirical evidence proving that Ada is being used successfully to implement large-scale AI systems. Congratulations to the participants for their AI with Ada successes!

Workshop Highlights

The SIGAda AIWG held a very busy and informative workshop in conjunction with the Summer '92 SIGAda meeting in Seattle, Washington. The workshop focus was Ada real-time and Artificial Intelligence (AI) issues. Software researchers and practitioners from the real-time and AI communities were brought together to exchange ideas and lessons learned. Most of our workshop time was devoted to presentations by those who have implemented large real-time AI with Ada systems. A summary of these AI with Ada applications is shown in figure 1 with the size in Thousands of Source Lines of Code (KSLOC) and the number of Knowledge Based Systems (KBS) rules.

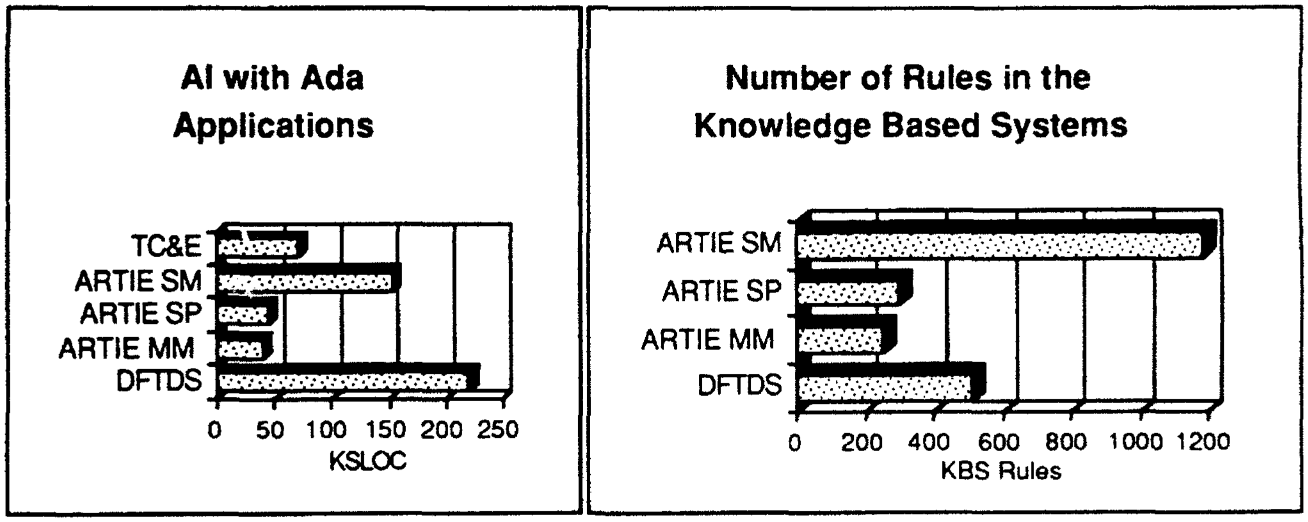


Figure 1: Summary of AI with Ada Applications

The acronyms used in the figure are Data Fusion Technology Demonstrator System (DFTDS), Ada Real-Time Inference Engine (ARTIE), tactical cockpit mission manager (ARTIE MM), search area planner (ARTIE SP), automated sensor manager (ARTIE SM), and Training Control and Evaluation (TC&E). These systems are a sampling of larger applications than most of those documented in the 1991 AIWG applications survey[1].

Workshop participants shared their experiences implementing large-scale real-time Knowledge-Based Systems (KBSs) in Ada. Participants from the Defense Research Agency (DRA) in the United Kingdom (UK) shared their experiences developing a real-time ship borne Command and Control system and their research activities in the area of validation and verification of safety-critical KBSs. A pair of related articles describing their activities, The Data Fusion Technology Demonstrator System (DFTDS) Project by John Miles and Validation and Verification of Safety Critical Knowledge-Based Systems by Jonathan Haugh are included in these proceedings. DFTDS is undergoing sea trials at the present time.

Boeing provided an update on the latest developments with their Ada Real-Time Inference Engine (ARTIE) and described some of their complex real-time reasoning based avionics applications including a tactical cockpit mission manager, a search area planner, and an automated sensor manager. George (Rick) Wilber and Robert Ensey provided a copy of their briefing slides which are enclosed in these proceedings as the article Embedded Real-Time Reasoning Concepts and Applications.

BBN has developed an AI with Ada application called Training Control & Evaluation (TC&E). Their experience provides some insight into rapid prototyping with Ada followed by the transition of a legacy prototype to Full Scale Engineering Development (FSED). Although Dan Massey was unable to attend the workshop, he provided an article which is titled Training Control & Evaluation (TC&E) for these proceedings.

Minutes of the workshop have been written in the form of a workshop discussion summary which is included with the Message from the Panel Chair for the Applications Experiences and Lessons Learned Panel. A workshop participant list follows this Executive Summary. Recommendations for specific AIWG actions are made at the end of this Executive Summary.

Issues

The panel discussed issues covering the complete software life cycle including the challenges of software engineering AI applications; the difficulties with AI requirements analysis; the legacy of an AI requirements analysis; testing, verification, and validation of AI applications; and maintenance for AI applications. Many of the issues and problems discussed during the workshop are applicable to AI applications developed with any language including the Ada programming language.

[1] J. Johns, June 1992, 1991 Annual Report for the ACM Special Interest Group for Ada Artificial Intelligence Working Group, MITRE Document M92B0000056 and the January/February issue of Ada Letters.

Software Engineering

AI applications typically begin with few documented requirements and are defined as well as developed with a series of evolutionary prototypes. This is the current state-of-the-art for AI applications design and development with Ada or any other programming language. Software engineering and Ada design methodologies typically begin with a set of well-defined, testable, verifiable requirements. Therefore, due to the lack of requirements, software engineering is a challenge for AI with Ada developers. Without a set of well-defined requirements, how do you "engineer" AI applications? This question is the subject of on-going debates and research. However, while there are no textbook solutions today, the information in these proceedings describes how AI with Ada practitioners are successfully "engineering" AI applications today.

AI Requirements Analysis

Requirements for AI applications are defined through iteration; that is, learning by doing. AI requirements are difficult, if not impossible, to specify without prototypes. Most of the effort for AI projects is spent defining requirements and prototyping. In the case of a 70,000 source line of code Ada application, 2/3 of the calendar time and labor were expended developing prototypes to define the requirements for the AI application.

AI requirements analysis requires a flexible process because defining AI requirements is an evolutionary process. This type of process *is not a fixed price problem*, and the typical use of DOD-STD-2167A may be too rigid for AI applications even with the software engineering discipline provided by the Ada programming language. We discussed many AI specific issues associated with the software engineering process and DOD-STD-2167A. These discussions led us to question whether AI requirements and design are the same as understood in the DOD-STD-2167A and other software engineering environments. Based on these discussions, the AIWG decided to be more active in expressing AI specific requirements and concerns to the standards bodies that are developing standards which impact the development of AI applications with the Ada programming language.

The Legacy of AI Requirements Analysis

Prototypes and human understanding of the problem domain are the standard legacy of an AI requirements analysis effort. What happens to this legacy? We would like to use our human understanding to specify the desired system in clear unambiguous requirements, but this is rarely possible. Prototypes typically reflect our understanding of how to implement a solution, and in many cases are required to prove that it is possible to "engineer" the full scale system. Can you successfully "re-engineer" a prototype into a supportable and maintainable system? Based on panel experiences described in these proceedings, Ada prototypes are being re-engineered into supportable and maintainable systems.

Testing, Verification, and Validation

The difficulties inherent in AI requirements analysis inevitably lead to problems with the testing, verification, and validation (V&V) of AI applications. Testing and validation activities ensure that the developed software system satisfies a well-defined set of requirements which unfortunately do not normally exist for AI applications. Verification activities ensure that the developed system is supportable and maintainable which raises issues associated with the feasibility of verifying non-deterministic systems that can learn and adapt over time. Testing and V&V are critical areas of current research because the public and the software engineering community have begun to focus attention on building trusted systems that are correct, dependable, robust, safety critical, efficient, and secure.

The panel discussions regarding testing and V&V led to two interesting thoughts. First, AI applications seek to emulate human intelligence and behavior. How do we test and validate humans? In general, the proof of human intelligence and their ability to learn is through on-the-job performance. Therefore, in essence, humans do not undergo the same scrutiny of test and V&V that we are trying to impose on AI applications. Second, instead of trying to perform an unnatural test and V&V scrutiny of AI applications, perhaps it would be more meaningful to develop techniques and tools that determine if an AI application is fit-for-purpose; that is, does it match the problem domain, is it operable in the proposed environments, and can it be learned with relative ease.

Maintenance

Maintenance is an area of critical concern as large-scale AI applications are fielded and must be supported for a long life cycle. Knowledged-base maintenance is also a critical concern during the iterative development of large knowledge-based systems. For an expert system, maintenance traditionally involves changes to the rule base as well as the facts which are normally considered data. The discussions in this area included questions such as "What is knowledge? Are rules considered data or software? Should facts in a knowledge-based system be treated as data or software?

During the panel discussions, several implementation techniques currently being used for expert systems were described as:

1. Implementing the rules in Ada for runtime performance.

2. Use two modes for the rule base: an interpretive mode for rule execution during development and a runtime mode that involves translating the rules to Ada for runtime performance.

3. A runtime mixture of interpreted rules and rules implemented in Ada.

One of the developers who is using the first approach of implementing the rules in Ada lamented about the tremendous overhead -- approximately 1 week -- to rebuild his real-time command and control system for any changes to the 510 rules in the rule base. The selected implementation techniques for an expert systems rule base influence both system maintainability and performance. Based on their development and maintenance experiences, the panel identified a critical need for a support environment that includes a real-time browser for runtime "peeks" into the system and knowledge-base maintenance tools.

Recommendations

Software engineering is a challenge for the AI community due to the evolutionary nature of AI applications. Software engineering is one of the strengths and advantages offered by a properly managed Ada environment. We, the AI with Ada community, should develop processes and tools that support the "engineering" of AI applications with Ada. The AIWG should add a section to the AIWG's annual report to describe software engineering processes for AI applications and assess the progress that has been made in the use of software engineering principles to develop AI applications. The AIWG should develop a database cataloging software engineering processes, tools, and applications with periodic publication of this information in Ada Letters.

In order to manage and engineer AI with Ada projects, the AIWG should establish a set of software metrics that are compatible with the evolutionary nature of AI applications. Further, the AIWG should conduct annual surveys to determine the current values for the software metrics so that this information can be used by the Ada community to manage and engineer AI applications. The software metrics should be published in the AIWG's annual report and Ada Letters.

The AIWG should become actively involved with standards bodies that are developing standards and other guidance that impact the development of AI applications with Ada. Of course, the first step in this process is to identify the AI specific requirements that need to be communicated to the standards bodies. The issues and problems discussed at this workshop should be matured into concrete requirements and recommendations for formal submittal to the appropriate standards bodies.

Many of the issues and problems faced by the AI with Ada community are the same problems faced by all AI researchers and developers. The AIWG should work closely with the AI community to concentrate our combined efforts on solving our common problems rather than focusing on the perceived differences between the AI and the Ada communities.

ACKNOWLEDGMENTS

I would like to thank all of the workshop attendees for their participation and contributions to the discussions and presentations that led to these proceedings. Jorge L. Diaz-Herrera gave an excellent presentation covering the current state-of-the-art in blackboard architectures. James Baldo coordinated the participation of our colleagues from the United Kingdom and led an interesting discussion about the challenging aspects of using Ada to implement real time systems. Rich Hilliard provided the AIWG workshop attendees an update on the activities of the Ada Run-Time Environment Working Group (ARTEWG). Henry Baker led the thought provoking discussions about Ada9X issues for the AIWG.

The workshop attendees extend a hearty "thank you" to Boeing for hosting a tour of their Boeing 757 testbed aircraft. While aboard the aircraft, we saw "AI with Ada in action" with demonstrations of Boeing's Ada Real Time Inference Engine (ARTIE) and the complex reasoning based avionics systems developed with ARTIE.

The MITRE Corporation has published these proceedings as part of their continued support for my activities as Vice-Chair of the ACM SIGAda AIWG. I want to thank Susan Maciorowski for her professional encouragement of my AIWG activities. Many thanks and words of appreciation go to my secretary, Kimberly Cole Taylor, who has worked very hard to put together this and other AIWG reports.

AIWG '92 Workshop Participants

Henry Baker
Nimble Computer
16231 Meadow Ridge Way
Encino, CA 91436
818-501-4956 (phone)
818-986-1360 (FAX)

James Baldo Jr.
Institute for Defense Analysis
1801 N. Beauregard St
Alexandria, VA 22311-1772
703-845-6624 (phone)
703-845-6848 (FAX)
baldo@ida.org

Mark R. Bowyer
DRA Farnborough Q153
Hants GU14 6TD, UK
252-24461 x2503 (phone)
252-377247 (FAX)
r_bowyer@uk.mod.hermes

Jorge L. Diaz-Herrera
SEI CMU (MSE project)
4500 Fifth Ave
Pittsburgh, PA 15213-3890
412-268-7636 (voice)
412-268-5758 (FAX)
jldh@sei.cmu.edu

Rob Ensey
Boeing Computer Services
Box 24346, Mailstop 9H-84
Seattle, WA 98124
206-394-3055 (phone)
206-394-3064 (FAX)
rob@patty.amas.ds.boeing.com

Jonathan Haugh
DRA Portsdown
Portsmouth Hants P06 4AA
Hampshire, UK
0705-333839
0705-333543 (FAX)

Rich Hilliard
Intermetrics
Cambridge, MA 08138
617-661-1840
rh@inmet.com

Janet Johns
MITRE Mailstop K203
202 Burlington Road
Bedford, MA 01730
617-271-8206 (phone)
617-271-2753 (FAX)
jfjohns@mitre.org

Mark Johnson
SMC/CNWS
Los Angeles AFB, CA 90009-2960
310-363-8770 (phone)
310-363-8725 (FAX)

Michael Looney
DRA Portsdown
Portsmouth Hants P06 4AA
Hampshire, UK
44-0705-332330 (phone)
44-0705-333543 (FAX)
MJL%ARE-PN.MOD.UK@Relay.MOD-UK

John Miles
DRA - Maritime Division
ARE Portsdown
Portsmouth Hampshire
P064AA England
0705-333839

Howard E. Neely III
Hughes Research Laboratories
3011 Malibu Canyon Rd
Mailstop MA/254/RL69
Malibu, CA 90265
310-317-5606 (phone)
310-317-5484 (FAX)
NEELY@MAXWELL.HRL.HAC.COM

John Tunnicliffe
DRA Portsdown
Portsmouth Hants P06 4AA
Hampshire, UK
0705-332002 (phone)
0705-333543 (FAX)

George (Rick) Wilbur
Boeing Computer Services
Box 24346, Mailstop 9H-84
Seattle, WA 98124
206-394-3055 (phone)
206-394-3064 (FAX)

Mik Yen
Boeing Defense & Space Group
Box 3707, MS4C-63
Seattle, WA 98124-2207
206-662-0213 (phone)
206-662-0115 (FAX)

TABLE OF CONTENTS

APPENDIX A

Welcome Message by Jorge L. Diaz-Herrera

Editor's Notes

Jorge L. Diaz-Herrera is the Chair of the ACM SIGAda AIWG. Jorge opened the AIWG Workshop with a welcome message describing the background and focus of both the AIWG and the workshop. In his welcome message, Jorge addressed several key issues with Ada, AI, real-time systems, and embedded systems.

SIGAda Summer Meeting June 1992 Artificial Intelligence WG

Critical Dependency of Embedded Systems on The Scaling up of AI Methods

Jorge L. Diaz-Herrera

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Notes

I would like to take this opportunity to introduce our working group and the overall theme of our workshop.

The working group was formed last year as a response to an increased awareness of the role played by AI in complex systems and specially in the areas of command and control

AI components will play major role in next generation of embedded systems.

DoD Software Technology Strategy (Draft, December 1991)

"... AI enables new and improved DoD mission functions by insertion of automated cognitive capabilities, e.g., smart weapons, smart sensors, or automated planning functions." (p. 2-44)

"There are three recurring themes adopted in DARPA and Service AI programs that are integral to the overall strategy: (1) the primary thrust of AI research, the invention of powerful new functionality; (2) the maturing of AI technology and its insertion into and integration with conventional software environments; (3) the impact AI technology can have upon conventional software engineering and the general software process ..."(p. 6-8)

Notes

Concomitant with the dynamic growth and increased popularity of AI and the progress seen in hardware technology is the vision that AI components will play a major role in complex, large, distributed and embedded systems.

This next generation of systems must be able to work together with more conventional software components performing traditional real-time tasks such as sensor reading and data fusion, control and scheduling functions, and actuator feedback processing.

They must also be well-engineered following established software engineering practices.

DoD Software Technology Strategy: Intelligent Systems

"For the purpose of this strategy, the term Intelligent Systems will refer to the Integration of AI methods, non-AI methods, and Innovative Software Engineering practices which synergistically enable powerful new functionality" (op. cit. p. 6.1)

Technical areas

- development of new intelligent functionality
- engineering issues of integration, verification, validation, real-time performance, and life-cycle maintenance.

1992 Jorge L Diaz-Herrera

Notes

The successful use of the AI-based Dynamic Analysis Replanning Tool (DART) for Desert Storm, AI techniques and environments are poised to support several of the primary Software Technology Strategy themes.

Within these technical areas, the emphasis is on the development of new intelligent functionality and the engineering issues of integration, verification, validation, real-time performance, and life-cycle maintenance.

according to the report (p 6-1), "SwTS addresses the requirements to develop new AI technology, evaluate that new AI technology in operationally relevant intelligent system experiments, and capture and deliver successful new AI technology in application-ready tools or frameworks ... Potential intelligent system projects are being formulated in each of the seven DoD science and technology thrust areas discussed in section 2.4)

Integration of AI Technology with more Conventional Software

Remains difficult

- implementation problems
- conceptual problems.

Implementation problems

- insertion of basic software engineering into AI
- re-engineering of existing AI components
- language issues

Conceptual problems

- responsiveness to rapidly changing data
- predictability of results
- fault-tolerance

1992 Jorge L Diaz-Herrera

Notes

The integration of AI methods and techniques with conventional software engineering remains difficult and poses both implementation as well as conceptual problems.

Implementation problems are associated with the insertion of basic AI technology into conventional software engineering practice and tools, and more specifically with an evaluation of the re-engineering *of existing AI components.*

Conceptual problems are best exemplified by the inability of AI approaches to respond rapidly and predictably to fast changing data *for controlling complex systems.* Current AI computational models are not based on a good model of behavior, from a resource utilization point of view, for a number of knowledge-based techniques such as searching, rule-based reasoning, semantic nets, etc.

AIWG Workshop SIGAda Summer'92

- AIWG workshop will build on the past 1989 and 1990 AIDA (AI and Ada) conferences which focused on Ada, Real-Time, and Artificial Intelligence (AI).
- Purpose is to bring together software researchers and practitioners from real-time, Ada and AI communities to exchange ideas, results, and lessons learned.
- The workshop will be composed of discussion groups working on different topic areas.
- Results of the workshop will be published as a proceedings by POET/IDA and as a SIGAda AIWG report.

Notes

Proceedings available from

George Mason University, Department of Computer science, Attention: Mary Lou Kiel

AIDA'86, November 12, 1986, Arlington VA.
Jorge L.Diaz-Herrera, Editor

AIDA'87, October 14-15, 1987, Herndon, VA.
Jorge L.Diaz-Herrera and Henry Hamburger, Editors

AIDA'88 November, 1988, Fairfax, VA
Jorge L.Diaz-Herrera and John Moore, Editors

AIDA'89 November, 16-17 1989, Fairfax, VA
Jorge L.Diaz-Herrera and Jan M. Zytkow, Editors

AIDA'90 November, 1990, Reston, VA
Jorge L.Diaz-Herrera, J.Baldo and David Littman, Editors

AIWG Workshop SIGAda Summer'92

Wednesday, June 24 1992:

1:45-3:00 pm -- Group chairs presentation

- Applications & research using blackboard architectures.
  Group Chair: Dr. Jorge Diaz-Herrera
- Ada and AI applications experiences & lessons learned.
  Group Chair: Janet Johns
- Real-time Ada development approaches and issues.
  Group Chair: James Baldo Jr.
- Ada 9X issues for AI systems.
  Group Chair: Dr. Henry Baker

7:00-10:00 pm -- Group organizational meeting.
Group chairs and participants discuss workshop details.

Notes

Each group chair will present a 15-minute overview.

- Applications & research using blackboard architectures: This group will assess state-of-the-practice of using blackboard technology for embedded AI systems based on current implementation experience and research that shows promise for near-term insertion, including an analysis of issues and limitations.
- Ada and AI applications experiences and lessons learned: This Group will focus on lessons learned through the experience of implementing AI applications using Ada. The Group will describe existing applications developed with Ada, identify Ada specific design and development issues, discuss Ada specific maintenance issues, discuss lessons learned for embedded and real-time expert systems, and assess current state-of-practice of Ada for AI systems. Other topics of discussion include knowledge base maintenance and tools for AI applications design and development.
- Real-time Ada development approaches and issues: This Group will identify the tools that are currently available to do Ada real-time development, identify real-time Ada issues and their impact to SDI, and assess Ada real-time development guidelines.
- Ada 9X issues for AI systems: This group will address AI issues and concerns with the proposed Ada9X standard. It will also lead a discussion on the adoption of an AI secondary standard proposal.

AIWG Workshop SIGAda Summer'92

Thursday, 25 June 1992

9:00am-5:00 pm: AIWG workshop.

Friday, 26 June 1992

1:30-3:00 pm: AIWG workshop findings

Each group char presents findings and conclusions from the workshop

[illegible]

Notes

Group chairs will prepare some discussion questions in advance for the various panels to use to keep the workshop informative and convey important points that the group would like to communicate with the audience.

Panel members will discuss their experiences and lessons learned, and the most pressing issues for the development of real-time embedded systems relevant to the topic area.

APPENDIX B

The Data Fusion Technology Demonstrator System (DFTDS) Project

Editor's Notes

This article is a treasure trove of information about the advantages of using Ada to implement large-scale real-time Knowledge-Based Systems (KBSs). The United Kingdom's Defence Research Agency (DRA) has implemented a large-scale real-time KBS for ship borne Command and Control. DFTDS has been implemented with 220 KSLOC of which 50 KSLOC implements KBSs for time-critical functions such as data fusion and situation assessment. DFTDS is a true KBS in Ada application as the rules are coded directly in Ada to achieve the run-time efficiency required by Command and Control applications which must process data from radar, sonar, navigation, electronic support measures (ESM), and other sensors. At the present time, DFTDS is installed on the Royal Naval Frigate HMS Marlborough and is undergoing sea trials.

This article discusses the lessons learned using Ada to implement DFTDS. Dr. Miles finds "Ada as a language has been found to be quite simple to use for a large real-time KBS and in some respects, notably run-time efficiency, abstraction, exception handling, strict typing, has distinct advantages over AI toolkits and expert system shells for engineered real-time applications." Prototyping to assess the potential of implementing time-critical functions such as those required for data fusion, situation assessment, planning, and reaction are discussed in this article. Dr. Miles also describes the DFTDS Ada shell and the use of a rule specification template with a semi-formal rule specification language to ensure a consistent style was used to specify the hypotheses and rules.

AI Topics: Blackboard architecture, expert system, data fusion, situation analysis, planning

Domain Area: Command and control

Language Interfaces: Unknown

Project Status: Undergoing sea trials and evaluation

Size of Ada Source Code: 220 KSLOC

Number of Rules in Knowledge-Based System: 510

Design/Development Methodology: Iterative prototyping

Hardware Platforms: MicroVAX 3800

THE DATA FUSION TECHNOLOGY DEMONSTRATOR SYSTEM (DFTDS) PROJECT.
(presented at the SIGAda meeting Ada/AI workshop, Seattle, Washington, June 1992)

John A H Miles
DRA Portsdown, Portsmouth, Hampshire, UK.

ABSTRACT

A large-scale real-time Knowledge-Based System (KBS) has been implemented entirely in Ada. The application is Data Fusion for shipborne Command and Control and the KBS is interfaced to several sensor systems, a relational database and provides five user workstations.

Ada was chosen for its real-time performance and its system engineering features. An Ada Shell using the Blackboard Model has been developed and a methodology for knowledge specification and implementation in Ada has evolved; these are described in the paper.

The real-time performance of the KBS has met its targets and few problems have been encountered with the Ada development system chosen. Experience with Ada, lessons learned and suggestions for Ada improvements for KBS are included.

1. INTRODUCTION

This paper describes a demonstrator project which resulted from a research programme to look at the application of AI, particularly KBS techniques, to command and control. Several laboratory prototypes were developed for functions such as Data Fusion, Situation Assessment, Planning and Reactive Resource Allocation. The success of these prototypes and particularly the potential of those addressing time-critical functions has led to a large-scale Data Fusion Technology Demonstrator System (DFTDS) being constructed and fitted to a warship for a two-year period of sea trials (reference [1]). The objectives of this programme are:

- to explore the use of KBS and other new technologies for providing automated support to Data Fusion and Situation Assessment functions
- to reduce the risk of procuring systems which use these technologies by examining all stages of the development, setting-to-work, performance and in-service support of a large-scale prototype

Both the conventional software and knowledge-based modules of the DFTDS have been successfully developed in Ada. All the modules including KBS operate in real-time, typically processing 100 messages per second. These features make the DFTDS a valuable, in some respects probably unique, source of data from a practical Ada/KBS application.

2. DFTDS DESCRIPTION

Figure 1 shows a block diagram of the main DFTDS modules. The Front-End Processor (FEP) receives real-time input messages from all the ship's sensor systems, from data communication links and from operators via the user interface module. Its main purpose is to present the Data Fusion module with messages in a suitable format for knowledge-based processing. The module provides filtering, re-formatting and recording of messages. It can also operate in a replay mode using data previously recorded or simulated scenario data.

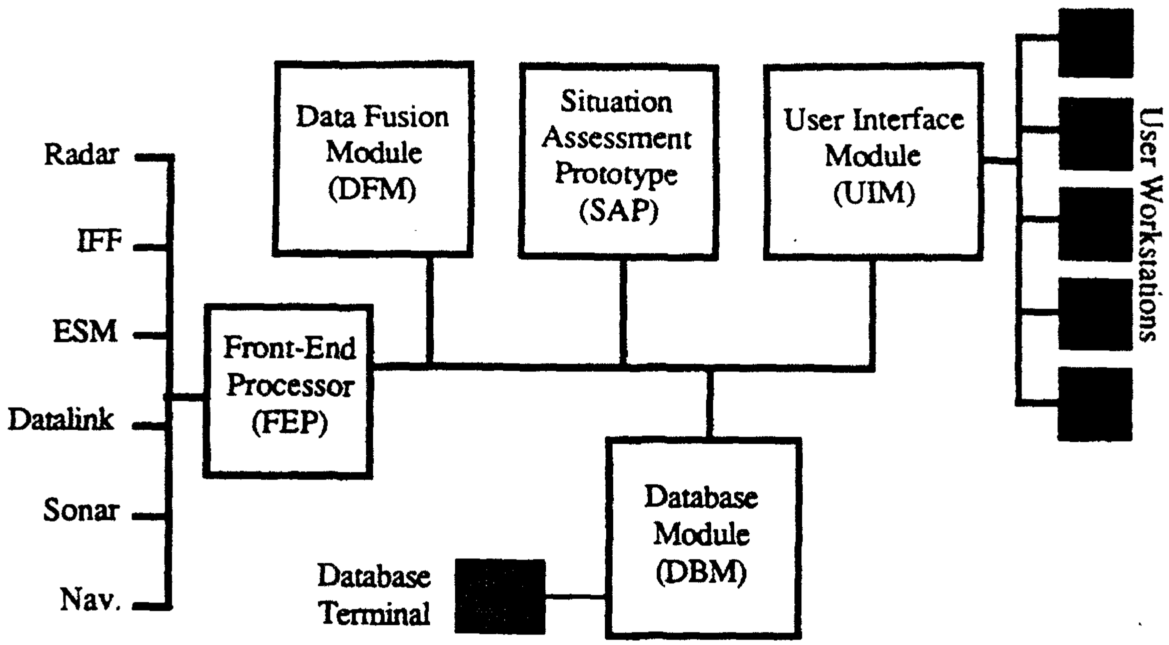


Figure 1 DFTDS Block Diagram of Modules

The Data Fusion Module (DFM) is the first and currently largest KBS module in the DFTDS. It is structured along the lines of a Blackboard System with rules, knowledge sources and a blackboard data structure entirely written in Ada. The purpose of the module is to correlate and combine data from all available sources (sensors and manual inputs) to form a fused tactical picture. It allows operators access to and interaction with each stage in the reasoning through the user interface module. Some of the reasoning in DFM requires access to the geographic and encyclopaedic databases held in the Database Module. Further details of DFM operation are given later.

The Database Module (DBM) provides geographic data (coastlines, navigation features, seabed features, shore features, air-lanes, etc.) for display and for answering on-line queries from DFM. Similarly, the encyclopaedic database in DBM, contains ship, aircraft, shore base, equipment, identity data etc., and provides query facilities for both DFM and for operators via the user interface module.

All user interaction with the DFTDS other than system control and database maintenance is carried out through the User Interface Module (UIM). This module provides a WIMP (Windows, Icons, Menus, Pointer) style interface on five colour graphics terminals and a form fill style interface for some data entry on two ordinary terminals. The windows on the graphics terminals give visibility of all stages in the data fusion process and allow the user to configure the display in any desired manner to suit his tasks.

The Situation Assessment prototype (SAP) is a second KBS module written in Ada using a similar framework to DFM. Its purpose is to take the detailed tactical picture produced by DFM and by applying further stages of reasoning construct a concise situation display showing the main groupings of objects and a prioritised threat table. All user interaction with this module is also through the UIM so that it appears as one system to the user even though it uses distributed KBSs. SAP will be enhanced in functionality in a second phase of the project.

All the DFTDS application modules communicate using the facilities provided by a System Support Module (SSM). This module, part of which resides with each application module, also provides control, timing, monitoring and fault reporting.

3. SIZE AND PERFORMANCE

The overall size of the Ada source code in the DFTDS is about 220,000 DSI (Deliverable Source Instructions). Of this about 50,000 DSI are KBSs. Other sizing metrics for individual modules are as follows:

DFM KBS:	
Input Message Types	100
Number of Rules	510
Number of Knowledge Sources	180
SAP KBS:	
Number of Rules	90
Number of Knowledge Sources	31
DBM (Encyclopaedic Database):	
Number of Tables	244
Number of Stored Procedures	112
Number of MMI Forms	68
Overall data size	15 Mbytes
DBM (Geographic Database):	
Number of Tables	12
Number of Queries	23
Overall data size	350 Mbytes
UIM:	
Number of windows	44
Number of window overlays	18

The main performance criteria for the DFTDS are:

Front-end maximum input message rate	180 messages/second
Maximum message rate input to DFM	100 messages/second
Maximum response delay in any module	1 second
Database queries	20 per second

4. HARDWARE AND SOFTWARE

The DFTDS runs on a network of five DEC Microvax[1] 3800 processors. Each main module (i.e. FEP, DFM, DBM, UIM, SAP) is allocated to a separate processor. This ensures that there is no contention for processing resource between modules. Ethernet with Decnet[1] software has been used for inter-processor communications. This has proved to have considerable overheads when used for high volumes of short messages and on some routes messages have had to be blocked into larger packets to achieve required throughput.

The DEC VMS[1] operating system and the DEC Ada[1] compiler have been used throughout. Apart from the SSM and the encyclopaedic database server each module is a single Ada program which runs as a single VMS process. DBM is written in Ada but uses a commercial Relational DBMS (Sybase[2]) for storing the encyclopaedic data and performing queries. The Ada part of DBM interfaces to the SQL routines using the Sybase provided Ada libraries.

[1] Microvax, Decnet, VMS, Dec Ada are trademarks of Digital Equipment Corporation.

[2] Sybase is a trademark of Sybase inc.

5. CHOICE OF ADA

Previous laboratory prototypes of the data fusion function had used a specially written blackboard framework, called MXA (reference [2]), because no suitable real-time KBS package could be found at that time (1983); MXA was based on the Pascal language. This framework provided some elaborate set intersection mechanisms but would only cope with simple scenarios in real-time. At the beginning of the DFTDS programme a method had to found to achieve real-time performance with the data rates found in a real operational setting.

A survey of available tools was carried out and three styles of implementation were benchmarked using a subset of a data fusion knowledge base with a test scenario. This benchmarking experiment is more fully reported and discussed in references [3] & [4]. Briefly it showed that a 'pure' production system implemented in a LISP machine environment was far too slow whereas a hybrid production system - procedural language based solution using an AI toolkit specially designed for embedded real-time applications produced better than real-time performance. Even though good performance was achieved with this toolkit, only a conventional language solution using a blackboard style framework actually achieved the required real-time performance. Figure 2 summarises the benchmark implementations and results.

System	Hardware	Time in minutes to process 30 minutes of data	Fraction of real-time
ART	Symbolics LISP machine	600	20.0
MUSE	Sun 4/260 workstation	15	0.5
Ada Shell	Microvax 3500 workstation	3	0.1

Figure 2 Benchmark Implementations and Results

Ada was chosen for the conventional language implementation because it is the latest language specifically designed for real-time applications and on the results of some very simple benchmarks seemed to have no significant performance penalties providing a good compiler was chosen. Various compiler/machine combinations were tested using the data fusion benchmark and the DEC Ada compiler running on a microvax proved to be the best combination for speed, support and reliability. These results were obtained in the first quarter of 1988.

6. KBS IMPLEMENTATION USING ADA

In order to produce the Ada data fusion benchmark a blackboard style framework had to be developed. This proved relatively simple for a small rule-set but it was recognised that a more engineered version with diagnostic facilities would be required to construct a large knowledge base and an 'Ada Shell' was produced. Initially an Ada tasking approach was taken to various functions of this shell but this was found to introduce run-time overheads owing to the high input message rates required. The final version has proved very efficient and has remained largely unchanged over the two and a half year DFTDS development programme.

A particular attribute of the Ada KBS approach taken is that the rules are coded directly in Ada making it a true KBS in Ada application. Apart from the excellent run-time efficiency that this approach gives it has also been found that the abstraction capability of Ada and the flexibility of a conventional language used in conjunction with a blackboard model, allows for an easier and better designed implementation than the more common production system approach. The latter is driven by the need for exact patterns to be specified for the implicit rule firing mechanism to work whereas the rule firing mechanism in the Ada blackboard approach, although requiring more hand-crafting, is much simpler, flexible and efficient and results in more concise rules.

A drawback in using Ada for the DFM is that a KBS demands a large amount of global data in the form of blackboard hypotheses, whereas Ada data handling concentrates on data abstraction and data hiding. The Ada compilation suite is not best suited to deal with this global data and this means that relatively minor changes to the blackboard specification result in prohibitively long re-compilation times - lasting an entire working day in some cases.

7. ADA SHELL

Figure 3 shows a diagram of the components of the Ada shell and the blackboard elements that are part of the application. In normal operation the Scheduler takes the highest priority Event off the Event Queue and calls the appropriate Knowledge Source(s) (if any). On completion of the Knowledge Source control returns to the Scheduler and the cycle repeats. The Scheduler also checks for input messages and for delayed Events as part of its cycle. A Clock task maintains time and an Input task places input events on the Event Queues.

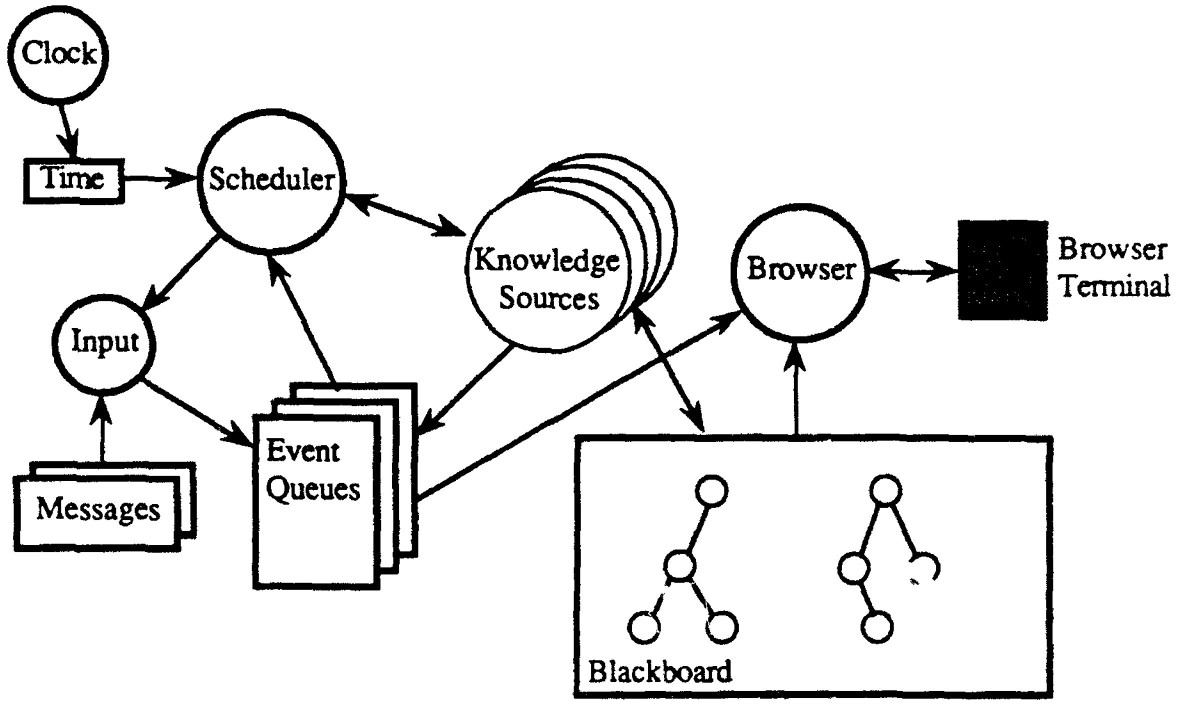


Figure 3 Ada Shell Components

For development purposes there is a Browser task which can be invoked by typing control-C on the Browser Terminal. This action transfers control to the Browser task and stops time and further events from being processed. The Browser has commands for displaying the contents of the Blackboard and Event Queues, for setting breaks on time/events and for stepping single events. It also allows events on the queues to be explicitly fired in any order, events to be removed and specific events to be inhibited or permitted. A DCL sub-process can be spawned from the Browser to provide access to the host operating system for viewing files, etc. With these simple facilities the action of each Knowledge Source can be closely monitored.

A blackboard system has a number of components and these are represented in Ada in the following manner:

The Blackboard - this is a data structure consisting of hypotheses and links. Hypotheses are implemented as record structures and these can be easily specified in Ada. Links are provided by set operations in a generic package. These operations allow links with a many-to-one, one-to-many and many-to-many relationships to be represented.

Rules - these are written as if...then..endif constructs within a procedure. The use of a procedure allows rules to be called from one or more knowledge sources. Although in principle similar to production rules, the Ada rules tend to contain more functionality and have more complex actions than those typically found in other KBSs. Output, in the form of messages to other modules, is generated directly by the rules. This output includes copies of blackboard data and explanations generated on request from the UIM.

Knowledge Sources - these are implemented as procedures which generally contain nothing but calls to rule procedures. In some cases it has proved convenient to provide some logic within knowledge sources for more 'intelligent' invokation of rules.

Events - there are three types of event; input events which have a type and an input message associated with them, normal events which have a type and a hypothesis reference associated with them and, delayed events which have a type, a hypothesis reference and a time associated with them.

The Ada shell operates as follows:

An input event is generated for every input message and there is a one-to-one correspondence between input event types and message types. Other events are generated explicitly by rules to indicate particular changes to the blackboard or to invoke processing at a later time. In theory there could be an event type for every type of change to every type of hypothesis but in practice events types are only created as required to suit the knowledge source and rule structure.

There are four normal event queues of different priority and the delayed event queue. The scheduler takes the highest priority event and using a look-up table calls the knowledge source applicable to the type of event. There can be more than one knowledge source assigned to an event or none at all; in practice there is usually just one since any rules that can potentially fire for the event can be put in a single knowledge source.

The event contains either a message or a hypothesis reference to which all of the rules called by the knowledge source have access. In other words the rules are directed at the change in the blackboard and they then apply their exact conditions to attempt to fire. This mechanism means that there is no guarantee that a rule called by a knowledge source will fire and consequently some time may be wasted evaluating rule conditions that fail but the event mechanism is so simple and efficient that it easily out-performs more rigorous approaches such as the RETE algorithm used in productions systems.

Rules from different functions can be called from the same knowledge source if their firing conditions are similar thus implementing the opportunistic nature of the blackboard model. It would be possible to implement more complex scheduling but the simple system described has been found quite adequate for the real-time data fusion application. It has sometimes proved useful to invoke a rule conditionally depending on the result of a previous rule but this can easily be implemented within the knowledge source.

8. KNOWLEDGE SPECIFICATION

The knowledge base is specified in terms of Hypotheses and Rules; definition of events and knowledge sources is considered to be part of the implementation process. Rules are written in a declarative style so that each one completely defines when and what processing will occur. A specification template has been designed (figure 4) together with a semi-formal language to ensure a consistent style of specification. The semi-formal language contains a range of keywords including 'if', 'and', 'actions', and a number of verbs for various types of action. Each rule is restricted at the top level to one or more 'anded' conditions followed by one or more actions. Conditions and actions may invoke further operations which can be of arbitrary complexity. These operations are specified with the rules in a pseudo-code form because they contain important domain knowledge. In other cases the rule specification may use a function or procedure in order to keep the specification concise but for which the details of operation are either self-explanatory or of a house-keeping nature unrelated to the domain knowledge; in these cases no further specification is given.

Rule No RXXXnnn <name>
(a) Summary
(b) Rule Specification

```
RULE RXXXnnn <name>
        if   <condition 1>
        and <condition 2>
        ....
        actions
        ....

END RULE RXXXnnn
```

(c) Assumptions/Limitations
(d) Definition of Criteria
(e) Supporting Operations
(f) Hypotheses
(g) Data Structures

Figure 4 Rule Specification Template

The summary (a) of each rule is an English version of the domain knowledge contained by the rule. The rule specification (b) is a more precise definition of what the rule does and it is required that the Ada code, although more detailed, follows the content and order of this specification, thus retaining the KBS philosophy of keeping specification and source code closely aligned. Part (c) is a comment heading under which any assumptions or limitations about the scope of the rule can be noted. Part (d) provides a 'definition of criteria'. This is used to reference more detailed specifications of any conditions used in the rule; Complex conditions or simply constants are generally expressed in meaningful words within the rule specification in order to keep it concise. Part (e) is intended for specifying any supporting mathematical operations used by the rule - in practice this is rarely used because such operations are usually not visible at the rule specification level and if they are, they are mostly self explanatory. Parts (f) and (g) list hypotheses and other blackboard data structures accessed by the rule. This was originally intended as a means to trace rule access and hence interaction between rules but such analysis has still to be performed.

Specification of the knowledge base, principally in terms of hypotheses, rules and operations, is contained in a document called the 'Acquired Knowledge-base Specification' or 'AKS'. This document is then used both for implementation in Ada and for validation of the knowledge-base by domain experts - it is the definitive reference for users, implementers and maintainers. Because the user interface to the KBS is also a very complex piece of software, a separate specification has been generated to detail every window and interaction with the user and knowledge-base.

9. DFTDS PROJECT STATUS

The main development phase of the DFTDS is complete and the system including the data fusion KBS has been integrated and set-to-work. Current activity is concerned with working-up the system onboard the trials ship and tuning the knowledge-base to optimise its performance against real data from ship's sensors.

A trials and evaluation programme has been formulated and has already commenced. It encompasses the following areas of investigation:

- KBS Performance
- Technology Issues
- Human Computer Interface (HCI)
- Manning and Personnel
- Value to command

KBS Performance involves the measurement of data fusion performance under a variety of normal and extreme operating conditions. This performance can then be compared with that of existing more manual systems, and the value and limitations of the KBS approach can be found.

Technology Issues include all technical aspects of development and in-service support; the use of Ada is one such topic.

The Human Computer Interface (HCI) has novel features for this type of shipborne system both in terms of its WIMP style and its KBS features such as explanations. Feedback from users who will be using the system under realistic conditions will be sought and will form valuable guidelines for future KBS procurements.

Manning and Personnel issues will be examined to determine the impact which the advanced level of automated support enabled by KBS technology has on the level of manning, the skills and knowledge required and the training requirements.

Value to Command addresses the impact which the quality of output from the DFTDS will have on the higher levels of command within the ship. If the DFTDS produces a more complete, accurate and timely tactical picture and assessment then this should result in improvements to the decision making process and hence to the whole command and control performance.

10. LESSONS LEARNED

At this stage in the programme the main lessons learned concern the development process. Those that relate to the use of Ada for KBS applications are as follows:

Knowledge Engineering - Ada as a language has been found to be quite simple to use for a large real-time KBS and in some respects, notably run-time efficiency, abstraction, exception handling, strict typing, has distinct advantages over AI toolkits and expert system shells for engineered real-time applications. The drawbacks to Ada are that does not directly provide the KBS styles of programming and these have to be hand-crafted before knowledge engineering can start. It might be possible, of course, to design a number of general purpose packages to suit a range of Ada KBS applications.

DEC Vax Ada Experience - this Ada system has proved extremely reliable with very few bugs or limitations being experienced throughout the DFTDS development. It does, however, require substantial computing resources to support the development teams and as the software has grown, the re-compile times have become very long (several hours). This seems to be due to contention for access to central libraries. No doubt other Ada systems have similar problems with large systems. It is a particular difficulty with a KBS as a considerable amount of iteration in developing and tuning the knowledge base is inevitable and indeed part of the development philosophy.

Performance - The run-time performance targets have been achieved though some optimisation has been carried out; the use of a run-time analysis tool (PCA) has proved very useful for identifying where most time is spent. Use of generics seemed to cause unexpected overheads and in some cases these have been removed. Run-time checking and exception handling has proven to be most useful in ensuring that the system continues running even when errors occur; the overheads in this checking seem surprisingly low.

Ada Shell Facilities - Although rudimentary, the facilities of the Ada shell have proved to be extremely useful for developing the data fusion KBS. The small units of processing (knowledge sources) and simple scheduling cycle coupled with the break, step and display facilities of the Browser provide good visibility of the program operation, though it is intrusive and non-real-time. In fact the facilities would be beneficial to any Ada program whether considered a KBS or not. Of course the facilities are not as sophisticated as an AI toolkit because the Ada language does not allow easy access to the program source code at run-time (though it can be used in conjunction with the source level Ada debug facility). Also, the display of blackboard data structures by the Browser has to be hand-crafted as the system is developed. If the Ada shell was integrated with the Ada debug tool then these drawbacks could be overcome.

Ada Language Facilities - Ada is obviously not an AI prototyping language but as a means of engineering a large real-time KBS application it has proved quite effective. A few features could be added to make AI programming easier, such as:

- Variables could have a 'nil' state; KBSs particularly deal with partial data sets and very large numbers of variables. To indicate whether a variable is set in Ada a further variable must be defined whereas in most AI languages a variable can have a 'nil' state to indicate it is unset.

- Objects; the DFTDS has used a rule-based approach throughout because rules are simple and reasonably predictable in processing terms which is important in a real-time system. However, it is recognised that Object Oriented Programme (OOP) is useful for some knowledge representation and has been used in some of our laboratory prototypes. Explicit support for OOP would be a useful addition to Ada.

- Type Hierarchies; Ada has a rather inflexible system of types, sub-types and discriminents. It should be possible to define a hierarchical data structure and to write packages or procedures with any required visibility of that structure. Currently, Ada requires code to have full downwards visibility of the structure even though it may only need visibility of one level. An example of this is where software is required to handle messages without needing to know their content. A similar problem arose in the design of the blackboard hypothesis structure where there are a number of different types of hypothesis but general purpose links are required between them. Eventually a single type of hypothesis was defined with discriminents for the contents.

- Atoms; For symbolic processing most AI languages have 'atoms' which are general purpose identifiers for arbitrarily complex structures. Ada has enumerated type variables, all possible values for which must be specified at compile time, and strings. Strings are cumbersome in a strictly typed language and are inefficient for processing purposes. The DFTDS implements an Ada interface to a relational database; the database holds identifiers as strings and the interface converts some of these into enumerated types, where they match types in the rest of the system, and some it keeps as strings. The problem is that the strings are unchecked and inefficient and the enumerated types are inflexible so that simple changes to the database require changes throughout the Ada code. There seems to be a need for a run-time enumerated type which is held internally as a number but appears as a symbol at input and output.

11. FUTURE WORK

As far as the DFTDS trials and evaluation programme is concerned the use of the Ada language, Ada environment and tools will be reported under the technology issues area. Further studies are planned into the knowledge engineering and knowledge base maintenance aspects of the DFTDS. These studies will formalise the methodology for using Ada for KBS which has evolved and will investigate tools for assisting the maintenance of a large knowledge base. Now that the DFTDS rule set contains 500 or more rules it is becoming increasingly difficult to maintain the specification and carry out verification of changes.

12. CONCLUSIONS

The DFTDS project has proved that Ada can be used effectively for large real-time KBS applications with very little additional support software.

A large knowledge base has been established and this is entering an evaluation phase which will report on all aspects of performance, procurement, support, HCI, manning and value to users.

The experience with the Ada products selected has been good and some suggestions for improvement to Ada for KBS have been put forward.

13. ACKNOWLEDGEMENTS.

The help of Dave Shaw and Mark Jeffard in checking and commenting on this paper is gratefully acknowledged.

This work has been carried out with the support of the Procurement Executive, Ministry of Defence, UK.

14. REFERENCES

[1] Byrne, C D, Miles, J A H, Lakin, W L, "Towards Knowledge-Based Naval Command Systems", pp 33-42, procs. of 3rd IEE C^3I Conference, Bournemouth, May 1989.

[2] Tailor, A, "MXA - A Blackboard Expert System Shell", pp315-333, Blackboard Systems, edited by Engelmore and Morgan, published by Addison-Wesley, 1988.

[3] Miles, J A H, Daniel, J W, Mulvaney, D J, "Real-Time Performance Comparison of a Knowledge-Based Data Fusion System using MUSE, ART and Ada", Presented at the Expert Systems Conference, Avignon, 1988.

[4] Miles, J A H, "Artificial Intelligence Applied to Data Fusion and Situation Assessment for Command and Control", Ph.D Thesis with Supplement, University of Southampton, 1988.

APPENDIX C

Validation and Verification of Safety Critical Knowledge-Based Systems

Editor's Notes

The United Kingdom's Defence Research Agency (DRA) has a three year program to study the validation and verification (V&V) of safety critical Knowledge-Based Systems (KBSs) with the use of domain dependent virtual machines. In the first year, a Data Fusion Language (DFL) was identified and formally defined and the requirements for a virtual machine to support the DFL were established. A virtual machine to support the DFL was designed and the design verified in the second year. A virtual machine prototype to support the DFL will be implemented in the third year. At the present time (year two), the virtual machine design is being verified.

This article provides some useful insight into the potential of domain dependent virtual machines for V&V of large-scale Ada KBSs. This approach is being tested with a subset of the Ada KBS which is currently installed on the Royal Naval Frigate HMS Marlborough and described in the associated article "The Data Fusion Technology Demonstrator System (DFTDS) Project".

DEFENCE
RESEARCH
AGENCY

Validation and Verification of 'Safety Critical' Knowledge-Based Systems

J.M.Haugh

This presentation concerned an investigation of the use of domain dependent virtual machines to aid the verification and validation processes carried out during the development process of a Knowledge-Based Systems (KBS). This virtual machine approach to KBS development may be summarised as: prototype KBS for solving one or more specific problems from a domain of interest; identify, from this prototyping exercise, a domain specific language (DSL) for expressing solutions to problems in this domain; implement a virtual machine that has this DSL as its source language; use this virtual machine to implement prototypes and deliverable KBS. The potential advantages of this approach include:

- much of the development process is brought into the realm of conventional software engineering;
- less distortion of domain knowledge than would arise if it was expressed in the knowledge representation language (KRL) of a development environment or KBS shell;
- satisfying engineering constraints, e.g. required speed of execution, target hardware, etc., can be addressed within virtual machine design, independently of particular knowledge representations.

Additional benefits may result if formal semantics are defined for the domain specific language and formal specification and development techniques are used to ensure that the virtual machine implements those semantics.The utility of a virtual machine approach is to be tested by the creation of a Data Fusion KBS, using a suitable subset of the knowledge-base of a large scale Knowledge-Based Sensor Data Fusion Technology Demonstrator System currently installed on a Royal Naval Frigate HMS Marlborough.

The work began in 1st June 1991, just over a year before this presentation and most of the work is being carried out at the Royal Military College of Science, Shrivenham, Swindon, Wiltshire, England. We are now at the stage of verifying our virtual machine design. By June 1993 we will have completed a review into the first two years of work. This review is intended to assess the viability of our original concept and perhaps redirect the final year of the investigation in the light of the problems we have encountered.

Developing Reliable KBS

Separate out those aspects of the system

that can be developed using conventional software engineering techniques

from

those that are particular to the knowledge-base approach.

Objective Of This Research

This research item investigates the effectiveness of formally engineering virtual machines to support application specific knowledge representation languages. The objectives of this item are:

- to being to bring as much possible of the development of KBS within the domain of conventional software engineering;

- to apply the software engineering techniques associated with 'Safety Critical' software to appropriate parts of KBS development.

- Thus to increase the effectiveness of verification and validation of such Knowledge-Based Systems.

Verification and Validation

Verification

- The use of formal specification techniques in the design and implementation of the virtual machine.
- The formal definition of the semantics of the supported application specific KRL.

Validation

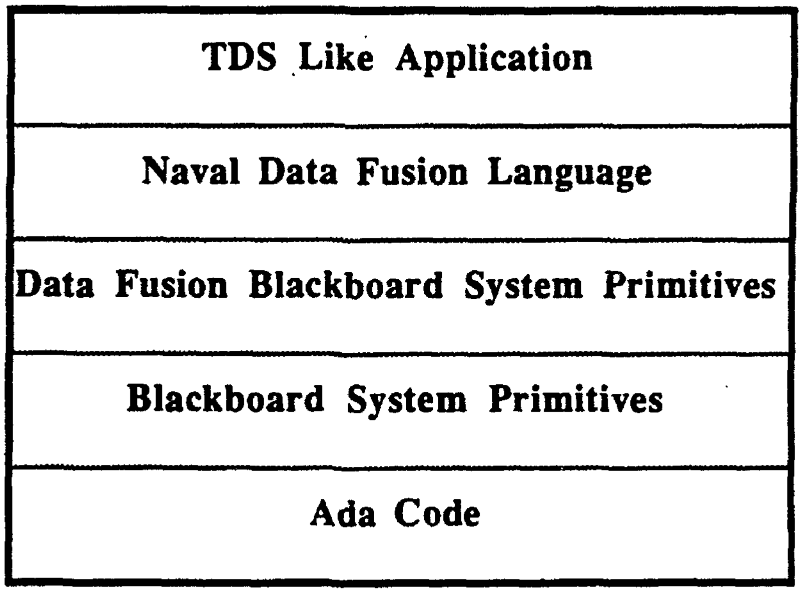
- The proof of properties of a system implemented in the application specific KRL that is simpler to perform than the direct formal transformation of a knowledge specification into an implementation.

Application Specific Knowledge Representation Languages (KRLs)

- The success of the virtual machine approach rests with their definition.
- They will imply one or more relevant knowledge-based system paradigms
- They will incorporate domain specific abstractions.

Multi Layered Virtual Machine

Naval Data Fusion Example


TDS Like Application
Naval Data Fusion Language
Data Fusion Blackboard System Primitives
Blackboard System Primitives
Ada Code

<u>“Validation & Verification of ‘safety critical’ KBS”</u>

- **Is being done under a 3 year contract by the Royal Military College of Science at Shrivenham.**

- **The work commenced on the 1st June 1991 and is, therefore, due for completion by the end of May 1994.**

Project Goals

Year 1:

to identify and formally define a Data Fusion Language for Naval Tactical Data Fusion, and to establish the requirements for a virtual machine to support this Data Fusion Language.

Year 2:

to produce a verified design, using appropriate structured and formal techniques, for the virtual machine to support the Data Fusion Language, and to animate the design to validate the Data Fusion Language.

Year 3:

to implement a prototype virtual machine to support the Data Fusion Language, and to evaluate its utility for Naval Tactical Data Fusion and the ability of the approach to provide high levels os assurance for knowledge-based systems for domains where reliability is regarded as a key requirement of those systems.

<u>First Year</u>
(June 1991-June 1992)

√ **Familiarisation** - with the ARE Technical Demonstrator Programme and Data Fusion Technical Demonstrator System.

√ **Review Formalisms** - VDM; Z; [B;] CSP [CCS;] OBJ [and similar methods] and Scott-Strachey denotational semantics.

√ **Specify Subset of TDS Knowledge-Base** - delivered in English, VDM and Ada form.

√ **Identify Data Fusion Language** - postulated in terms of a BNF syntax.

→ **Define Formal Semantics of the Data Fusion Language** - taking slightly longer than anticipated but a complete definition will be provided - almost complete.

× **Investigate Non-functional Requirements** - postponed after an initial look.

Definition of Data Fusion Language Formal Semantics

- No one formalism sufficient for defining the DFL semantics.
- VDM for the semantics of the parts of the DFL used to define the blackboard structure.
- CSP for the semantics of the parts of the DFL used to define the control structures.
- Validity of the particular VDM/CSP mix will be demonstrated.

Investigation of Non-functional Requirements

Not fully investigated because:

- Priority given to the definition of DFL semantics;
- Available time limited;
- Initial investigation indicated problem even more difficult than originally seemed.

Second Year

- **Identify Virtual Machine Design Approach** - an investigation of structured and formal methods to see which are applicable and the tool support which is available.

- **Design Virtual Machine** - design will comprise an architectural model and a formal specification defining that model.

- **Verify Virtual Machine Design** - it is unlikely that full formal verification can be brought to a logical conclusion within the cost constraints of the project. However it is hoped that sufficient progress will be made to establish what may be achievable in this area.

- **Animate Virtual Machine Design** - by translating the formal specification for the virtual design into an executable language.

- **Review Data Fusion Language** - so that amendments to its definition can be made in the light of the design, design validation and design animation of the virtual machine.

Third Year

- **Identify Virtual Machine Implementation** - a review of languages and tools suitable for the implementation of the virtual machine on an available hardware such as a Vax or Sun.

- **Implement Virtual Machine** - testing will be based upon conformance to the formal semantics of the Data Fusion Language and the virtual machine design (this will be the largest single task of the project).

- **Evaluate Virtual Machine** - In order to evaluate the utility of the virtual machine approach, a different subset of the TDS knowledge-base will be identified and translated into the Data Fusion Language - a final report on the utility of this virtual machine approach will be generated bringing this contract to a close.

End Products

- The Virtual Machine Prototype.
- Assessment Of The Virtual Machine Approach.

APPENDIX D

Embedded Real-Time Reasoning Concepts and Applications

Editor's Notes

Boeing has successfully implemented several intelligent real-time embedded avionics systems with the Ada programming language. These systems include a Tactical Cockpit Mission Manager (40 KSLOC of Ada, 250 rules), a Search Area Planner Tool (45 KSLOC of Ada, 300 rules), and an Automated Sensor Manager (153 KSLOC of Ada, 1,181 rules). This information is in these proceedings, but I wanted to emphasize that these applications are operational AI with Ada applications totaling of 238 KSLOC that implements 1,731 rules of complex reasoning based avionics systems.

Boeing developed these applications with an innovative approach that combines the "engineering" rigor of conventional software development with AI rapid prototyping techniques. Ada Real-Time Inference Engine (ARTIE) is a tool which provides an interpretive development environment and a small, fast embeddable inference engine for runtime performance. ARTIE offers an innovative approach to rapidly prototyping embedded real-time reasoning software with an interpretative development mode and an automatic code generator for generating embedded real-time code for execution on the target platform. Workshop participants were in mutual agreement that interpretive tools such as ARTIE are a valuable asset for real-time embedded systems prototyping, development, and debugging.

AI Topics: Cooperating expert systems, forward chaining, iteratively recursive inferencing

Domain Area: Intelligent Avionics Systems

Language Interfaces: A Pascal inference engine was developed in 1987.

Project Status: Test and evaluation

Size of Ada Source Code: ARTIE is 25 KSLOC with associated tools and the embedded inference engine is less than 3 KSLOC; Avionics reasoning based applications total 238 KSLOC

Number of Rules in Knowledge-Based System(s): 1,731 rules in 3 Ada applications

Design/Development Methodology: Iterative prototyping

Hardware Platforms: Apollo 3000, 4000, and 590; Sun 2/60, 3/60, and 4/60; Digital Equipment MicroVAX and VAX; Silicon Graphics IRIS

EMBEDDED REAL-TIME REASONING CONCEPTS & APPLICATIONS

Rick Wilber
&
Robert Ensey

Boeing Defense & Space Group
Seattle, Wa (206) 394-3055

Agenda

Overview Embedded Real-Time Reasoning?

- Real-Time Embedded Reasoning Systems **Concepts**
- Knowledge Driven Processing **Methodology**
- Knowledge Based Software **Development Process**
- Ada Real-Time Inference Engine (ARTIE) **Tools**

Applications of Embedded Real-Time Reasoning.

- On-Board Mission Manager (OBMM)
- Tactical Mission Manager
- Search Area Planner Tool (SAPT)
- Real-Time Route Planner

Automated Sensor Manager Overview

Current Status and Plans

Embedded Real-Time Reasoning Systems

Why?

- Modern Combat Missions Require Complex Avionics Control Systems
- Vast Quantities of Mission Data Must be Managed
- Robust Software for Mission Flexibility is Required
- Current Systems Provide Very Limited Reasoning Capabilities
- Factory Automation Demands Real-Time Reasoning and Control

What?

- Embedded Real-Time Reasoning Systems Attributes (Definition):
  - Perform Embedded Reasoning and Decision Making
  - Provide Real-Time Performance
  - Use Fully Engineered Development Methods (Ada etc.)
  - Effectively Combine Standard Engineering and Reasoning Functions
- Provide Efficient Iterative Software Development Capabilities

How?

- Use Hybrid Artificial Intelligence (Knowledge Based) and Conventional Software Development and Processing Techniques
- Use Iterative Development Process: *Concepts - Prototypes - Applications*

Embedded Real-Time Reasoning Systems Concepts

Conventional Software Tools Provide:

- Structured Engineering Environments
- Language Constructs for Algorithmic Processing: Sequential & Conditional Execution, Iteration, Algorithm Invocation, Etc.
- Real-Time Performance in Embedded Environments

Conventional Software Development Tools **Do Not** Provide the Constructs Necessary to Code Complex Logic Structures

Current Artificial Intelligence (Knowledge Based) Systems Provide:

- Efficient Methods for Representing Logic (Knowledge)
- Automated Reasoning and Decision Making Techniques

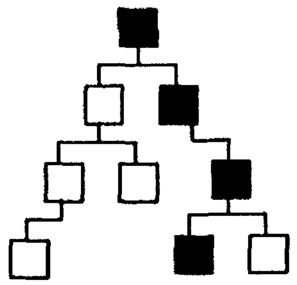
Current Knowledge Based Systems Usually:

- Do Not Operate in Real-time or in Embedded Environments
- Lack Engineering Rigor
- Require Specialized Computing Platforms, Languages & Engineers

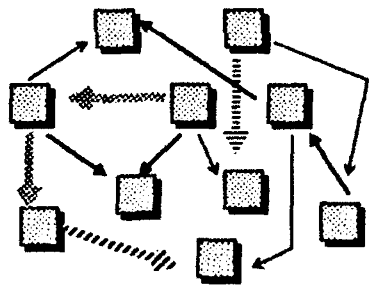
Embedded Reasoning vs. Conventional Systems

- Embedded Reasoning Systems Differ from Conventional Systems Because they Perform Autonomous Reasoning and Decision Making to Control Software Execution
- Their Basis of Decision Making (Logic) is often *ad-hoc*, User/Expert Defined.

Conventional Logic: Tree

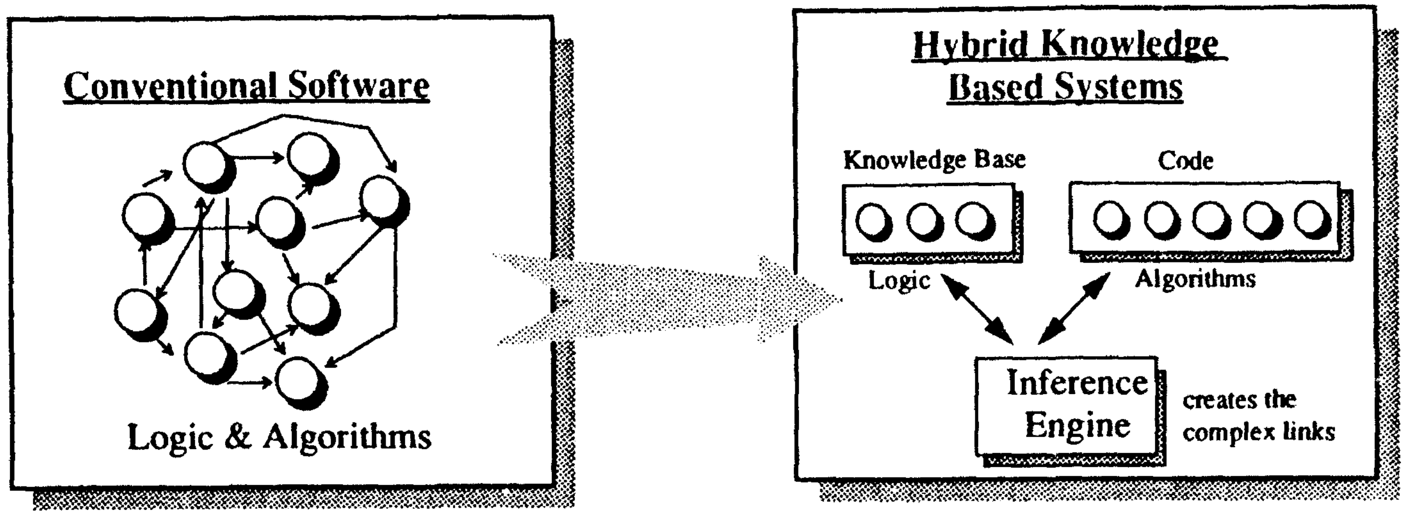


Complex Logic: Network



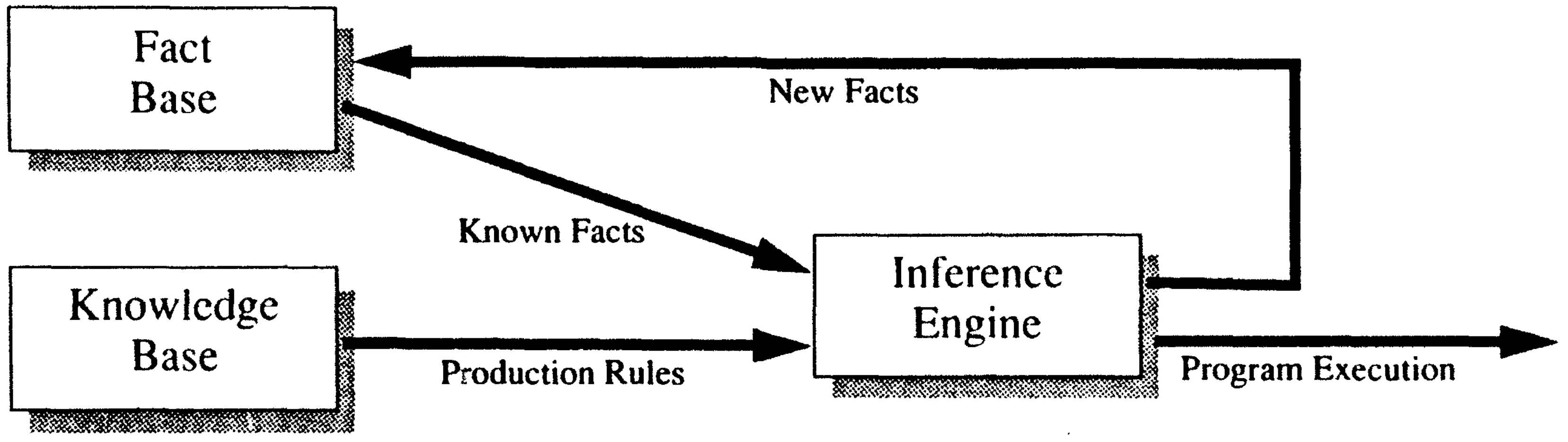
- Complex Control Logic Networks are Difficult to Represent using Conventional Techniques
- These Networks are Coded as Conditionals with Numerous Flags and Switches which are Difficult to Develop, Understand and Maintain

Knowledge Based Approach



- Separates the Control Logic from the Algorithmic Code
- The Algorithmic Code Becomes Less Complex, More Modular and Discrete
- The logic is Expressed at a Higher Level of Abstraction in the Knowledge Base Similar to Relations in Relational Databases
- Let Inference Engine Create the Complex Logic Code

Knowledge Base Inference Methodology



Production Rules

- Knowledge Stored as If-Then Statements that Define Fact Relationships

Inferencing

- Combining Known Facts and Knowledge to Infer New Facts
- Iteration Creates Automated Chaining

Knowledge Driven

- Fact Assertions Automatically Focuses & Drives Appropriate Code Execution

Sample Knowledge Base

```
RULES
  Rule_1
    IF     a new Popup_threat & Currently on a Bomb_Run
    THEN   UPDATE Countermeasure to Jam

  Rule_2
    IF     a new Popup_threat & Currently on a Navigation_Leg
    THEN   UPDATE Planned Altitude to Minimum
           UPDATE Countermeasure to Avoid_Threat

  Rule_3
    IF     Countermeasure is Avoid_Threat & Fuel is Abundant
    THEN   CALL the Route_Planner
```

Knowledge Based Software Development

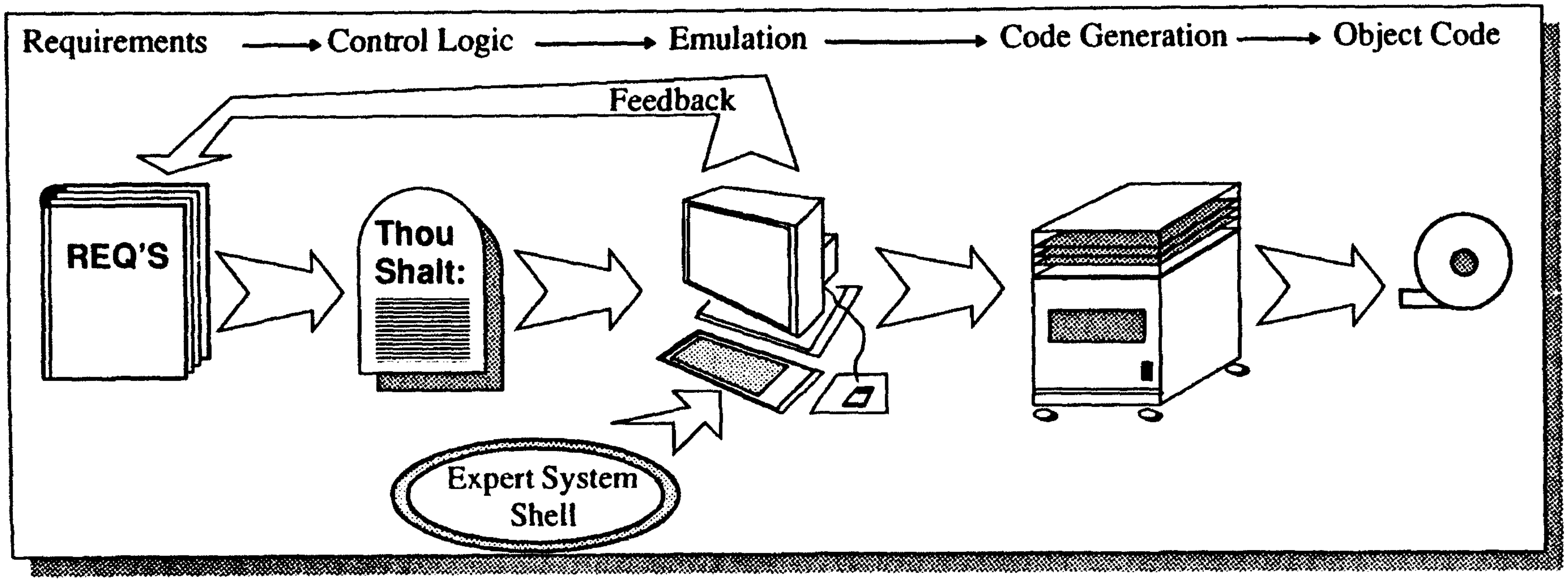
Rapid Prototyping

- Complex Logic Systems Must Be Developed Iteratively Due To Changing Requirements And Evolving Domain Knowledge
- Efficient Iterative Software Development Requires Rapid Prototyping Techniques
- Interpreters Allow Rapid Prototyping But are Inefficient at Run-Time
  - Support Rapid On-line Logic Modification
  - Greatly Reduce the Time Spent Performing Recompilations

Auto Code Generation

- Auto Code Generation Allows The Reduction (Elimination) Of Symbolic Processing
  - Increases Execution Performance and Reduces Code Size
  - Reduces the Run-Time Inference Engine to a Simple Data Base Access System
  - Converts Logic Directly to Ada Procedures
  - Does Not Support Rapid Prototyping

Knowledge Based Software Development


Requirements → Control Logic → Emulation → Code Generation → Object Code
Feedback
REQ'S
Thou Shalt:
Expert System Shell

Knowledge Based Software Development Uses an Interpreter for Early Development and a Code Generator for the Final Product

Advantages of Knowledge Based Software Development

- Early Viewing of Code Performance Through Emulation & Auto Code Generation
- Supports Tighter Coupling Between the Coders and Engineers
- Allows Shorter Development Time
- Eliminates AI Code From the Final Product

Ada Real-Time Inference Engine (ARTIE) Tool

ARTIE is the Main Tool that Facilitates Embedded Reasoning Software Development

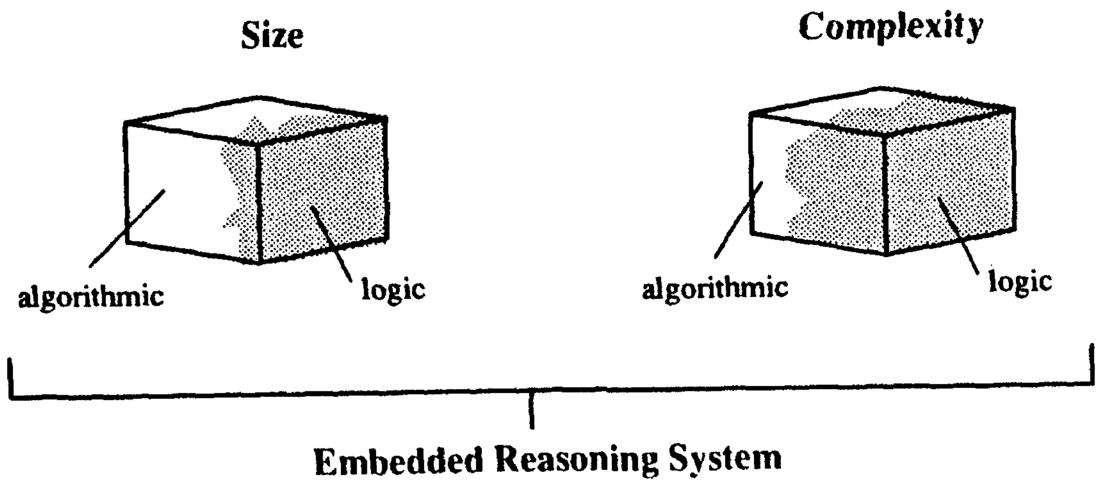
- It is Boeing Developed (IR&D)
- Supports Rapid Prototyping Through an Interpreter
- Provides Embedded Real-Time Execution Through Automatic Code Generation
  - Rules To Procedures
  - Control Logic To Table Lookups

ARTIE is:

- Small - Embedded Version is Less Than 3,000 Lines Of Code Overhead
- Fast - 8000 Rules/Sec On Sun SPARC 2 Interpreted, 14,000 With Code Generator
- Embeddable And Portable (Ada Code)
- Supports Cooperative Expert Systems
- Provides Modularity Through Multiple Knowledge Bases
- Includes Various Levels Of Debug And Tracing Capabilities
- ARTIE with all associated tools (parser, interactive user interface, debug/ monitoring tools, etc.) 25 KLOC

Complexity: Logic vs. Algorithmic Code

- Complex Reasoning Based Avionics Systems Contain large Amounts of Logic or Knowledge
- Most of the Complexity is Contained in the Logic
- Most of the Volume is Algorithmic Code and is Controlled by the Complex logic



Technology Development Time Lines

MILESTONES	1987	1988	1989	1990	1991	1992
On-Board Mission Mgmt Concepts (OBMM)	Apollo	Parallel & Embedded				
Heuristic Search Based Route Planning	Pascal	Ada	RCS Search			
Pascal Based Expert System Technology	Pascal Infer Eng	ARTIE				
OBMM Demonstrations		B-1B Mock-up	720-B Flight Tst			
Ada Real-Time Inference Engine (ARTIE) Developed		Interpreter	Code Generator	Embedded & Distributed		
Knowledge Based Software			Concepts			
Tactical Mission Manager			Devel			
Search Area Planner Tool			Planner	User Interface		
Automated Sensor Manager				Proto	Development	757 Flight
Real Time Black Board					Demo	Ada & ARTIE
Sensor Fusion					Devel	Flight Test Contract

Demonstrated Technology Applications

On Board Mission Manager (45 kloc, 125 Rules)

- Automated Mission Management for Strategic Bomber Missions
- Automated Real-Time Mission Planning (In-flight)
- Intelligent Battle Management Interface
- Advanced Threat Recognition and Countermeasure Selection
- Real-Time Distributed Reasoning & Processing* (Pascal Inference Engine)
- Flight Test and Aircraft & Systems Integration (Flight Test 1989)

Demonstrated Technology Applications

Tactical Cockpit Mission Manager (40 kloc, 250 Rules)

- Automated Mission Management for Tactical Missions (Ground Attack)
- Advanced Mission Planning
    - Flight Path Selection
    - Tactics Planning (Pop-to-Dive, Toss, Level)
    - Weapons Selection (Mk-82 & 2000# (Dumb, Hi-Drag & LGB), Maverick)
- Automated Sensor Control Concepts (FLIR, Maverick)
- Ada (ARTIE) with Portability (Apollo, Sun, Vax, Silicon Graphics)

Demonstrated Technology Applications

Search Area Planner Tool (45kloc, 300 Rules)

- Contract to Develop Ground Based Relocatable Target (RT) Mission Planner
- All Ada and ARTIE Runs on a Sun Workstation
- Adds Area Limitation Data Base Evaluation to the Real-Time Route Planner
- User Friendly Operator Interface
- Real-Time Performance (Develop 250 Mile Flight Path 10 Seconds)

Automated Sensor Manager Overview

• Sensor Manager	31 KLOC,	411 Rules
• Sensor Planner	45 KLOC,	35 Rules
• FLIR Expert	20 KLOC,	20 Rules
• Man Machine Interface	35 KLOC,	710 Rules
• Data Manager	22 KLOC,	5 Rules
• TOTAL	153 KLOC,	1_181 Rules

Automated Sensor Manager Overview

OBJECTIVE:

- Affect (Semi)Autonomous Real-Time In-flight Search, Detection, and Identification of Mobile and Relocatable Targets
- using Multiple Sensors
- and an Automated Sensor Manager operating both Autonomously or Manually.

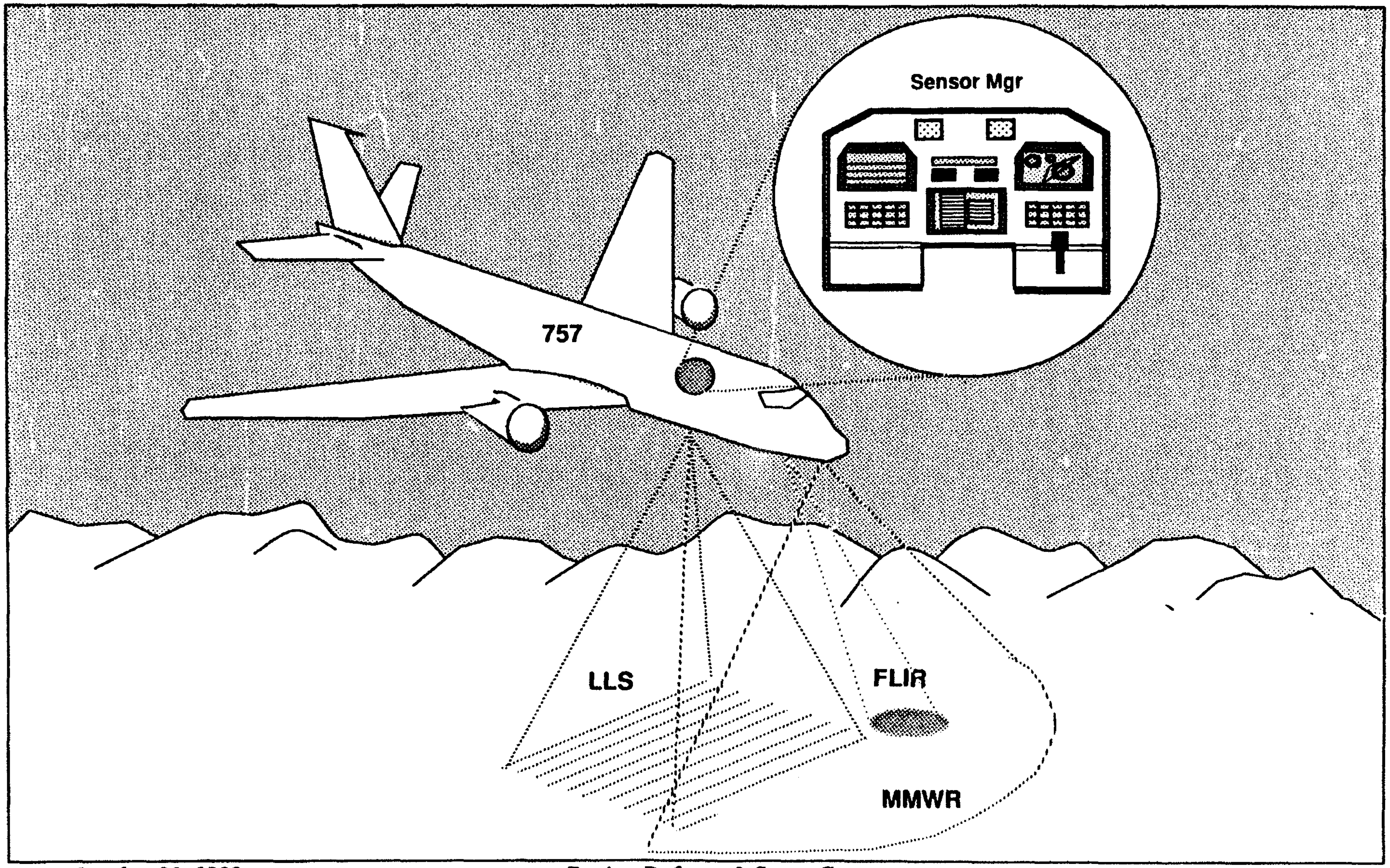
Automated Sensor Manager Computing Technology is a Natural Follow-on to Automated Mission Management

This is Our Most Recent Focus for Technology Application

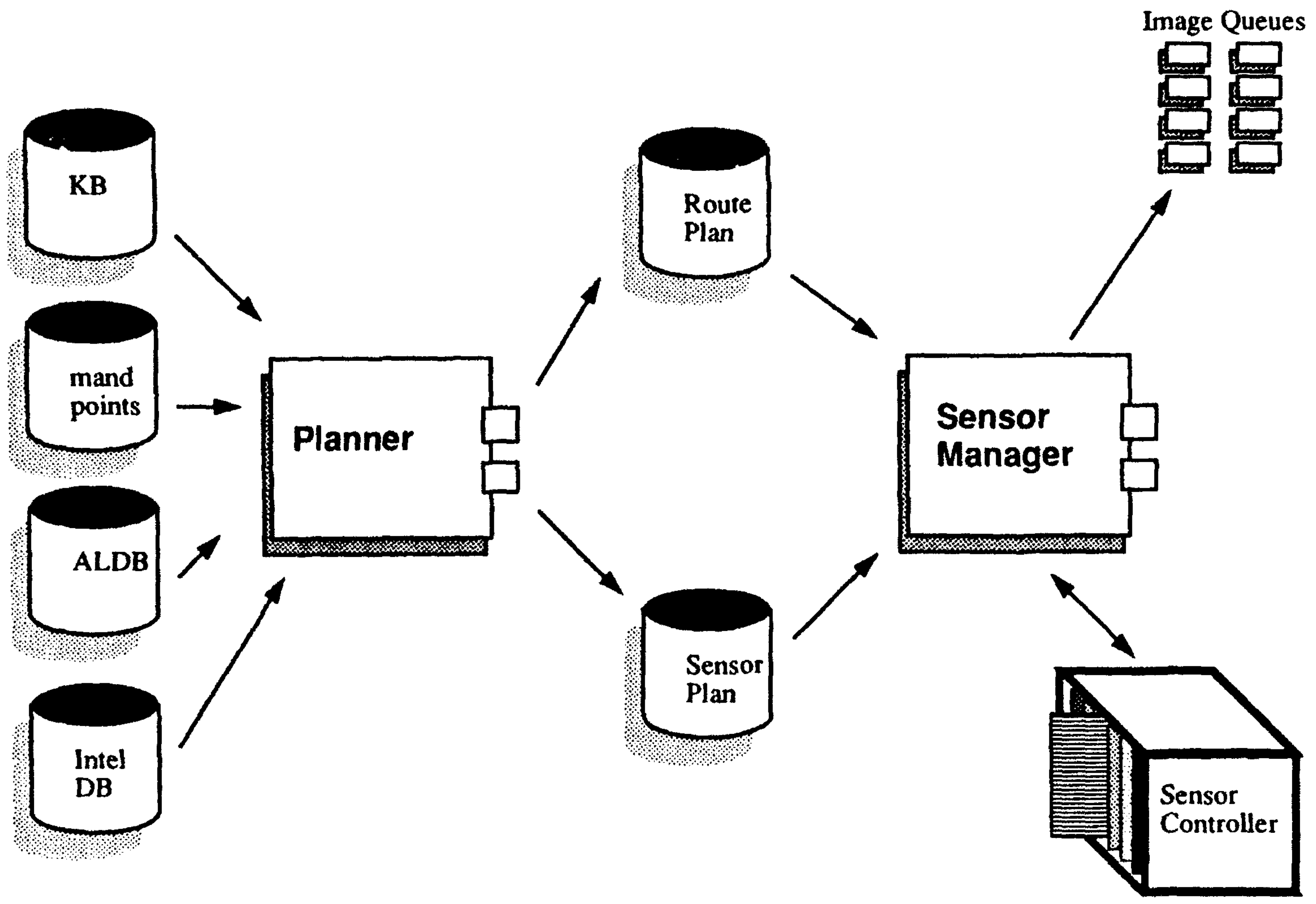
Technical Focuses:

- Sensor Control and Sensor Data Fusion
- Expand Mission and Sensor Planning
- Real-Time Distributed Reasoning and Processing
- Flight Integration and Demonstration
- Advanced Man-Machine Interfaces

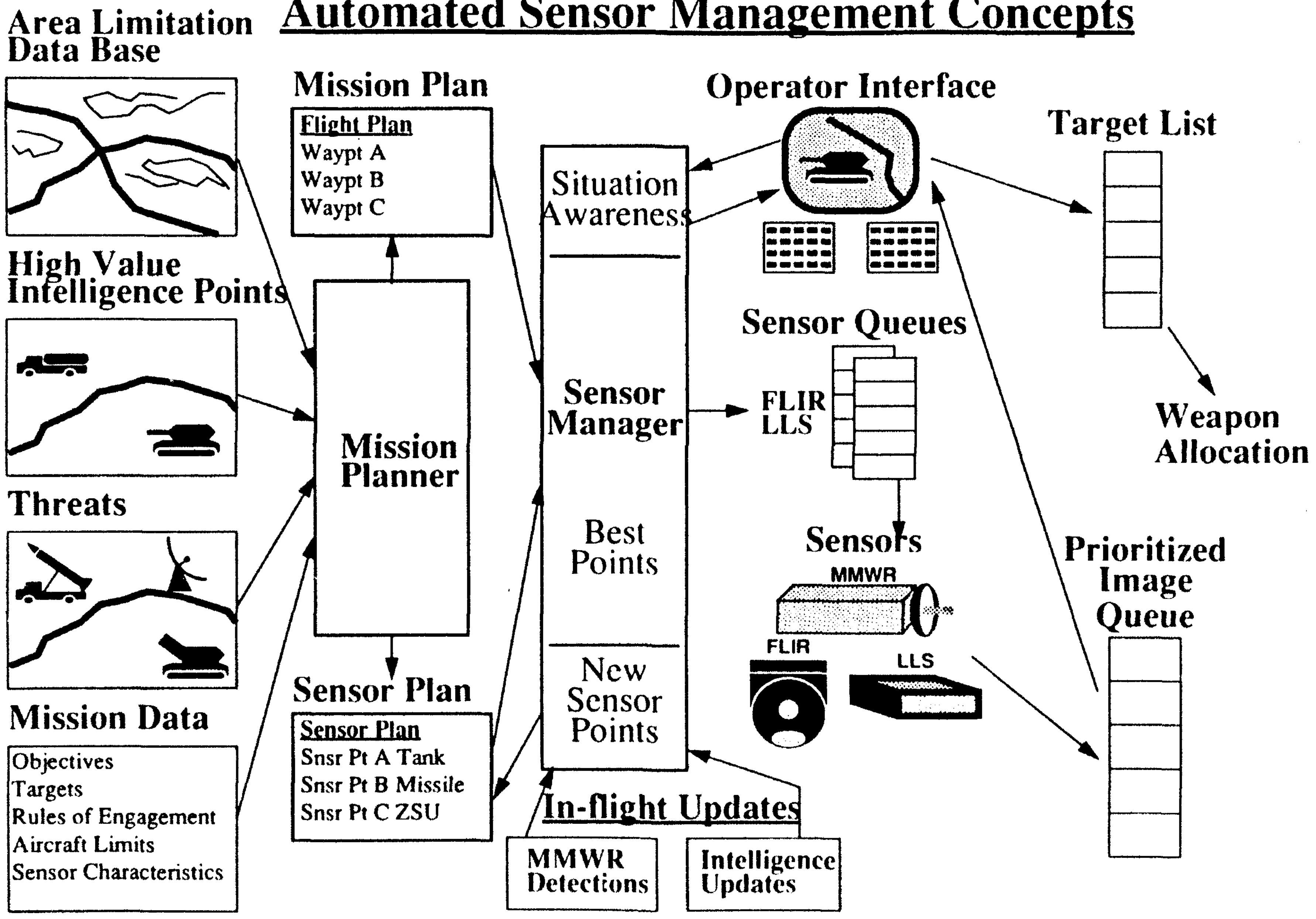
Automated Sensor Manager Overview



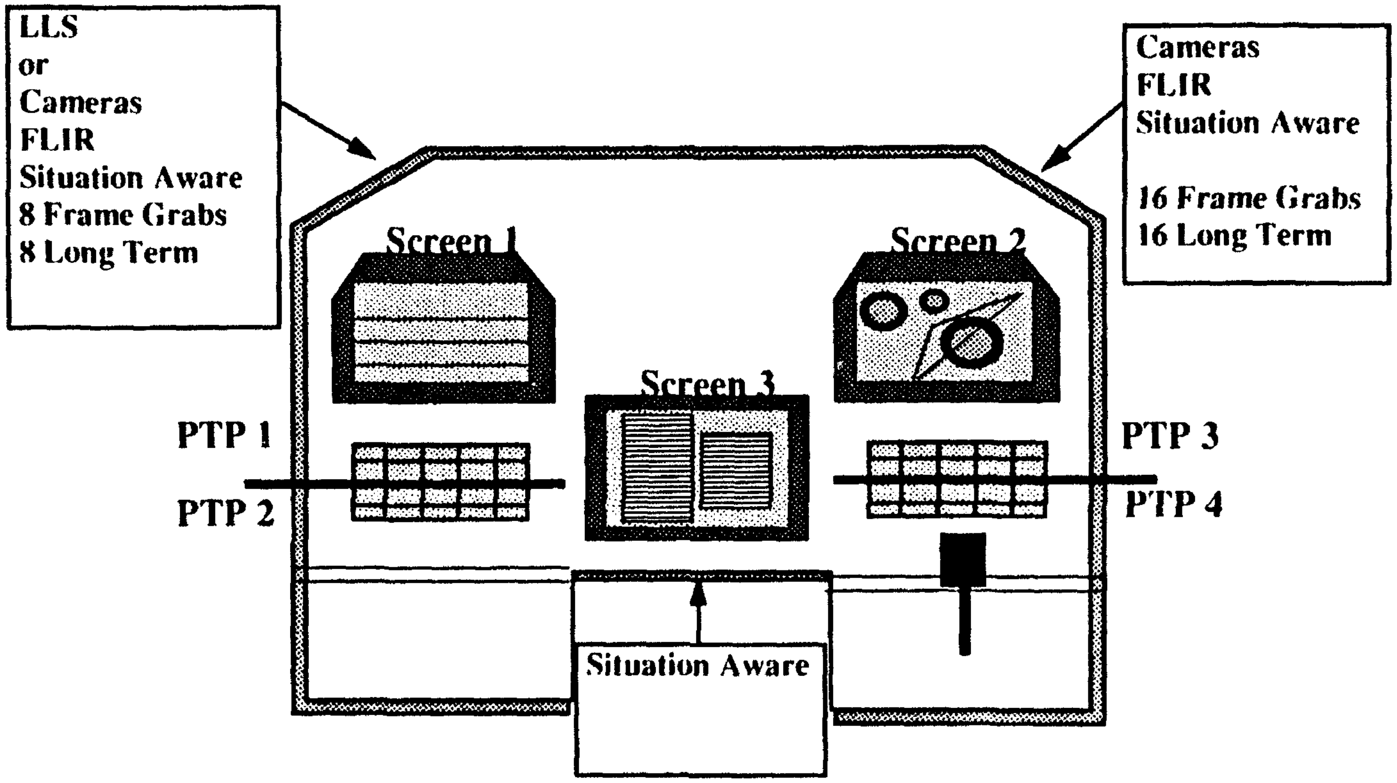
Automated Sensor Manager Overview



Automated Sensor Management Concepts

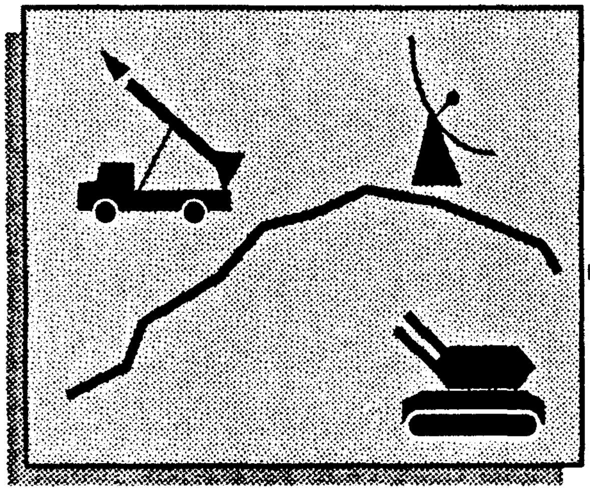


Operator Controls

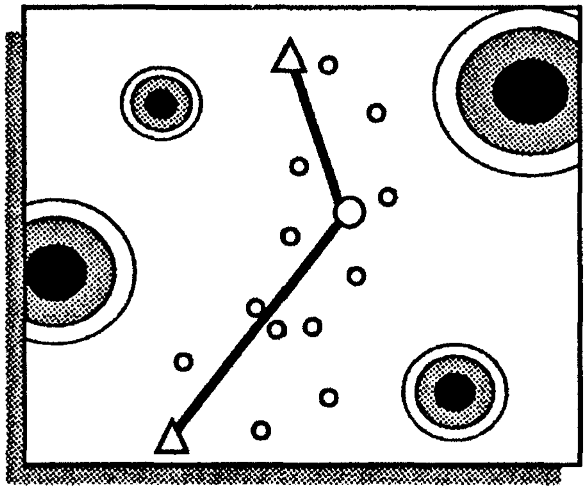


- Supports Multiple Color Graphic Displays
- Uses Programmable Keyboard Inputs
- Provides Facilities for Storage and Retrieval of Multiple Freeze Frame Sensor Images

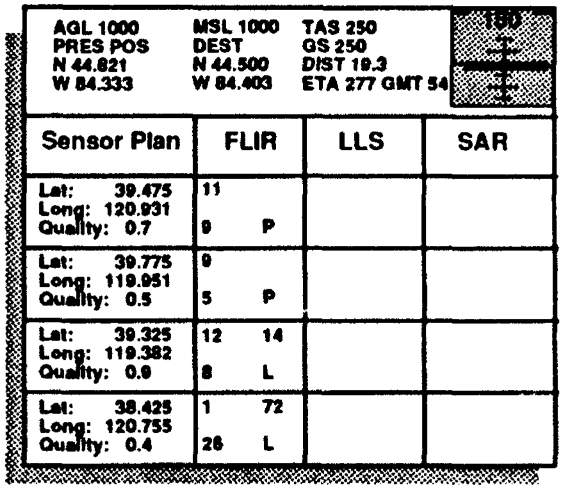
Sensor Manager Displays



Sensor Images



Situation Formats



AGL 1000
PRES POS
N 44.821
W 84.333

MSL 1000
DEST
N 44.500
W 84.403

TAS 250
GS 250
DIST 19.3
ETA 277 GMT 54

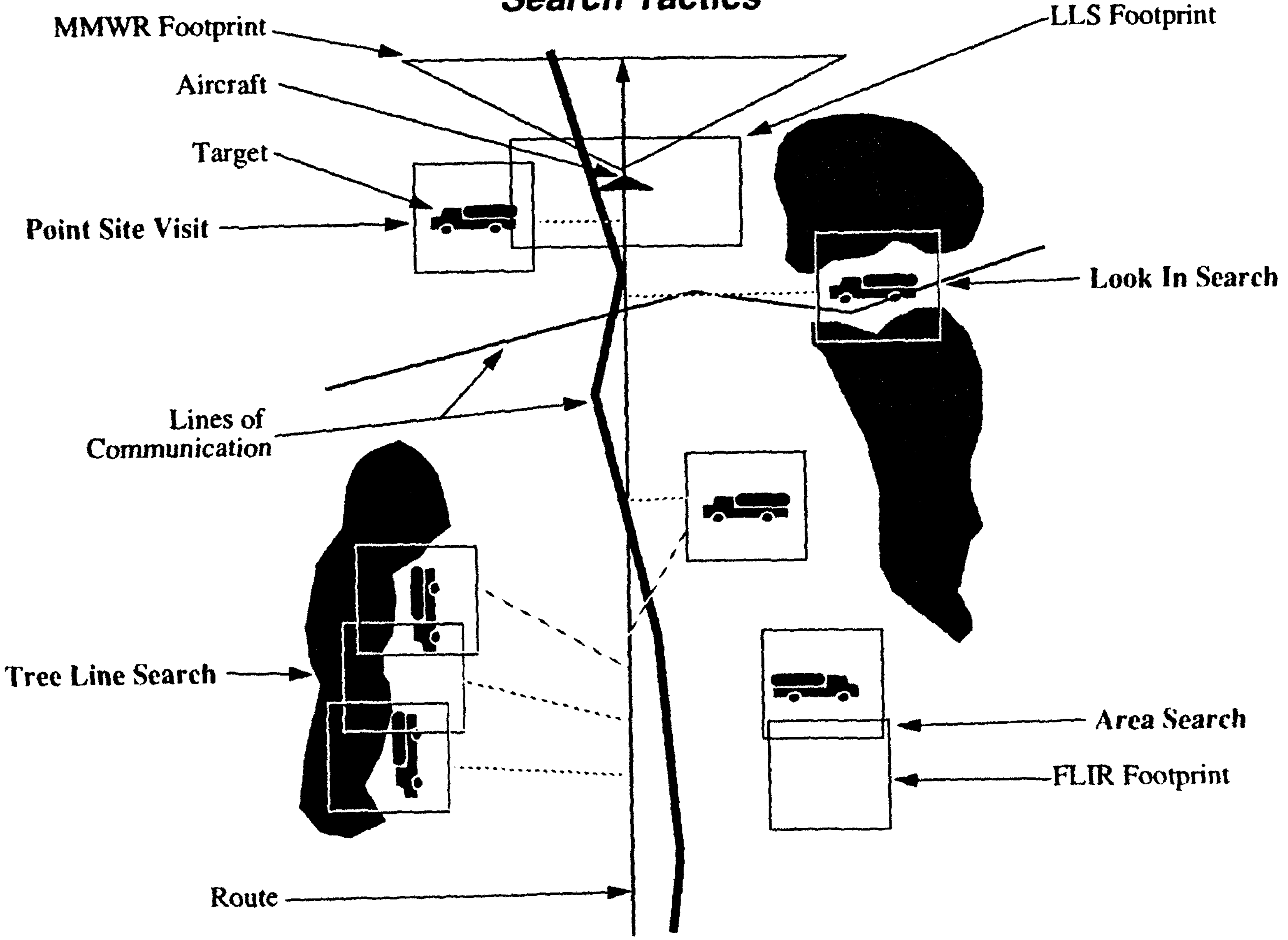
Sensor Plan	FLIR	LLS	SAR
Lat: 39.475 Long: 120.931 Quality: 0.7	11 9 P		
Lat: 39.775 Long: 119.951 Quality: 0.5	9 5 P		
Lat: 39.325 Long: 119.382 Quality: 0.9	12 14 8 L		
Lat: 38.425 Long: 120.755 Quality: 0.4	1 72 28 L		

Sensor Manager Data

Three Classes of Sensor Manager Display

- Sensor Images with Graphics Overlays
- Situation Awareness Displays with Color Graphic/Icons for God's Eye
- Sensor Manager Reasoning & Targeting Displays

Search Tactics



Real-Time Embedded Reasoning Status

DemonstratedAccomplishments:

- In-Flight Real-Time Embedded Reasoning in Ada Based Reasoning
- Efficient Knowledge Based Development Techniques
  - Rapid Prototyping Using an Interpreter
  - Auto Code Generation For High Speed Processing
- Effective Hybrid AI and Conventional Processing
- Parallel Distributed Processing and Reasoning (RTBB, Merit, Motorola, Sun)
- Real-Time Embedded Mission Planning (Route, Sensor, Tactics, Weapons)

Demonstrated On Board Mission Manager in Flight (Route Planning, Threat Id, Mission Management)

Delivered Search Area Planner Tool to USAF for Automated SRT Planning

Integrated Tactical Mission Manager with Off-The-Shelf Tools and Simulators

Successful RTBB/ARTIE Integration for Full System Reasoning

Ongoing Activities

Continue Refining and Applying Embedded Reasoning Concepts

- Past Efforts Developed Prototype Tools Only
- Support Current User Base (Contracts, C4I,, AWACS, RTBB, CMT Avionics, ESD-PD, Tech Staff Flight Controls Branch, B-1B Sensor Manager)

Refine Logic Control Concepts

- Look at Broader Avionics Control Problems
- Adapt Reasoning (Control Logic) Tools to New Problem Scope (Objects, Relational Data Base, Robust Ada Types)
- Develop More Portable (Ada C++) Distributed Systems
- Finish Integration with RTBB to Provide Full System Reasoning (Application, Resources, Fault Tolerance, Inter-Process Communication & Reasoning)
- Improve Robustness of Core Tools (ARTIE, Planner, Simulation)
- Finish Code Generator Package (Documentation, Packages)
- Further Refine and Evaluate Large Scale Software Development

Exploit the Technology in Depth on Current Projects

APPENDIX E

Training Control & Evaluation (TC&E)

Editor's Notes

BBN has successfully used an iterative prototyping approach to perform requirements analysis, design, and Full-Scale Engineering Development (FSED) for a 70 KSLOC Artificial Intelligence (AI) application. This application, Training Control & Evaluation (TC&E), is a valuable example of the success that can be achieved by rapid prototyping AI applications with Ada. One of the most difficult tasks in a rapid prototyping requirements analysis activity is the transition of the legacy prototypes and knowledge to the FSED. The TC&E experience offers some valuable insight into a successful transition of prototypes from the requirements analysis to a FSED for a DOD-STD-2167A project.

AI Topics: Blackboard architecture, goal-setting, path-following, progress tracking, obstacle avoidance, target selection

Domain Area: Training, simulation and modeling

Language Interfaces: C and 4GL

Project Status: Fielded

Size of Ada Source Code: 70 KSLOC

Design/Development Methodology: Iterative, rapid prototyping

Hardware Platforms: Sun

Training Control & Evaluation (TC&E)

Dan Massey

BBN
Cambridge, Massachusetts

617-873-3515

BBN recently completed development of a major distributed simulation system (called AGPT) for the German Army. This system was built in Ada, and includes a group of CSCIs (abo · 70 KSLOC of Ada) which support the instructor's interface to the system and provide interactive control over simulated elements of a virtual environment which are endowed with the ability to adapt and react to a changing tactical situation in an "intelligent" manner. This part of AGPT, called the Training Control & Evaluation (TC&E) segment, is based on a segment of the SIMNET system previously implemented in the US for DARPA.

The original SIMNET implementation (called Semi-Automated Forces, or SAF) was built as a prototype in LISP on Symbolics workstations and gradually ported to C over a number of years. The present implementation of SAF is about 300 KSLOC. These program sizes are not comparable since TC&E includes functionality not present in SAF, and vice-versa. In addition, about 65 KSLOC of C libraries (counted in the SAF total) are used in TC&E to support certain system housekeeping functions (network connections, matrix and quaternion math, real-time terrain analysis) that were common to both systems and for which assured consistency of implementation was deemed to be very important.

Although the "higher-level cognitive functions" simulated in TC&E are entirely written in Ada, including such functions as goal-setting, path-following, progress tracking, obstacle avoidance, target selection, etc., and involve the systematic application of a set of rules (some implemented explicitly as Ada procedures and others provided in a more general rule definition language), the ability of the system to operate in real time does depend on the fact that the most computationally intensive parts of terrain analysis (computing the lines of visibility surrounding a point in the virtual world) are implemented in carefully crafted and tightly coded C.

Most of TC&E Ada (about 80%) was built bottom-up, prior to agreement on either a system specification or a definite detailed design. Essentially, the core of TC&E is a rapid prototype, written in Ada, which has been subsequently refined through a series of incrementally enhanced releases. During the early phases of development (prior to agreement on the system specification) a number of pre-release versions of parts of TC&E, up to about 50% functionality, were demonstrated and evaluated in depth with the client to help in finalizing the specification.

The TC&E prototype (about 80% functionality) became an operational definition of segment design that was used in the full-scale engineering development of the production code. Through an agreed process of CSU documentation and test, large parts of the prototype code were formally "adopted" into the final product. Other parts were significantly reworked to improve testability, reusability, and reliability before adoption. Only about 20% of the TC&E segment code was produced de novo during FSED. These parts, largely enhancements to the prototype backbone, were specified, designed, and documented through a relatively formal engineering change process.

We consider that the entire prototype development phase of TC&E (which represented more than two thirds the calendar time and labor to obtain the final product) was a requirements analysis activity (in the sense of Royce's water-fall life cycle) and covered the inner cycles of a spiral development process (in the sense of Boehm).

TC&E Acronym List

CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DARPA	Defense Advanced Research Agency
FSED	Full Scale Engineering Development
KSLOC	Thousands of Source Lines of Code
SAF	Semi-Automated Forces
SIMNET	Simulation Network
TC&E	Training Control & Evaluation

APPENDIX F

Ada 9X Issues for AI Systems

Editor's Notes

Henry Baker chaired this panel at the workshop and led some very thought provoking discussions about Ada 9X issues for AI. Henry describes the classic implementation characteristics of traditional AI programs -- late binding, large address spaces, extensive use of shared memory for communication, extremely large programs, very dynamic problem spaces, complex and non-homogeneous data structures -- and offers suggestions for an AI Annex that includes procedures as parameters, a pragma Garbage_Collection, provides more general type initialization, generic types, and type instantiation parameters. His issues paper and briefing are titled Ada9X Issues for AI Systems.

Issues paper for Ada/AI/RT WG Workshop, Summer '92 SigAda Meeting, June 24-25, 1992.

Ada9X Issues for AI Systems

Henry G. Baker

Nimble Computer Corporation, 16231 Meadow Ridge Way, Encino, CA 91436
(818) 501-4956 (818) 986-1360 (FAX)

We must offer suggestions to the Ada-9X Committee for allowing/enhancing AI programs in Ada.

Defense AI Applications in Ada-83? In Ada-9X? What are the issues?

Can AI programs be successfully written and deployed in Ada-83? Do the changes contemplated in Ada-9X make the writing and deployment of AI programs any easier? Do there still exist major "gotchas" in Ada-9X which will seriously decrease the performance and/or increase the cost of developing and deploying AI programs in Ada?

AI programs are some of the largest *programs* around, in terms of lines of code, complexity, cost, etc. I.e., they are large *programs*, but small amounts of *data*, relative to more traditional embedded systems programs. E.g., *compiled* program text of large AI system can take 20 megabytes, whereas "large" non-AI programs use 1-2 megabytes for both program and data (not counting large passive databases such as terrain mapping databases).

What are the design, implementation and maintenance implications of such large programs in an Ada environment? E.g., small changes may cause massive recompilations—it could take hours or days to "make" a new version of a system. With a program of this size, is it *ever* "delivered"? If only 10^{-3} of such a program is changed annually, this may still be 10,000 lines of code changed per year. Rumors exist of massive software changes/fixes for the Patriot missile system while the unit was already in battle. Significant changes may be required in a "Pilot's Associate" program *for every mission.* In other words, "program" may have become "data", to be loaded for each and every mission. Does this mean that "dynamic loading" a la Berkeley Unix or Unix V Rel. 4 should become part of an Ada run-time system? Does this require a "persistent Ada heap", a la current "object-oriented databases"?

A "signature" characteristic of AI programs is their "late binding" of control constructs, which are universally implemented by means of first-class function closures. These closures are dynamically constructed functions which can be passed as arguments, returned as values, and stored into data structures as values. Ada-83 was expressly forbidden by its Steelman Requirements to have no such capability. Ada-83 offers *generic* functions and procedures, which can emulate some, but not all, of these late binding constructs. Do the capabilities of Ada-9X provide enough relief to satisfy the AI developer, or should we send the Ada-9X team back to the drawing board?

AI people have been requesting garbage collection for Ada at least since 1980 [Schwartz80], yet no vendor provides it, and Ada compiler/runtime validation does not require garbage collection. Yet GC is an extremely valuable tool in allowing the decomposition of large systems without increasing the probability of failure due to *dangling references*. Such dangling references are becoming more and more likely with the dramatic increase in pointer-based programs due to the popularity of "object-oriented" programming. Can garbage collection be emulated on top of Ada with enough efficiency to support the heavy computational demands of AI programs?

Traditional AI systems require a large address space and the shared-memory paradigm. Yet many embedded systems are designed with hardware that supports a distributed-memory/message-passing model, and it may be quite difficult to map AI programs onto these platforms. The Ada parallel process model clearly prefers an explicit exchange of information via the rendezvous mechanism, and only grudgingly supports the notion of asynchronous access to shared data. Yet the most popular model for parallel, embedded real-time AI systems is the "blackboard" model, which has at its core a database shared and asynchronously updated by all processes!

Although Ada was standardized in 1983, production quality compilers were not available until the 1986-87 time frame, and significant bugs are still prevalent in Ada83 compilers today. For example, generics could not be used reliably in the first generation of Ada83 compilers, and "storage leaks" continue even in today's Ada83 runtime systems. Thus, it seems prudent to recognize that it may be 1995 before debugged, reliable compilers and runtime systems are available for Ada-9X. In this case, another generation of weapons systems will be developed in Ada83—i.e., they will not be able to take advantage of any of the newer Ada-9X capabilities. Since AI capabilities are being put into systems today, what is the near-term effect of doing this in Ada83 v. Ada-9X? What are the long-term effects—i.e., efficiency, maintenance, obsolete protocols, etc.—of this delay?

Some "100% Ada" projects are using Ada as "just another language", to be loaded into a separate address space on a classical operating system with multiple address spaces. Any synchronization between the separate address spaces is implemented by means of non-Ada capabilities—e.g., locking in a "file" system. The Ada strong typing system is side-stepped by reading and writing to external "files". Worst of all, Ada run-time checks within an "application" (i.e., address space) are disabled, with any protection provided by the hardware—e.g., "bus error". Do such system designs conform to the spirit, as well as the letter, of the Ada law, or are they a pragmatic solution to an inflexible language standard? Perhaps such systems recognize the inevitable need to interface Ada with COTS technologies—most likely C, C++ or even Lisp(!).

Issues of ancillary standards and tools. Are the MIL-STD's for software design, documentation, implementation and testing appropriate for AI programs, or are they too rigid? Can the complex notions of AI programs even be expressed in these "design methodologies" and "design tools", or are new methodologies and tools required. Do the proposed documentation and coding standards put the AI programmer into a straight-jacket (assuming that Ada itself hasn't already)? Are the APSE/KAPSE/... tools part of the solution, or part of the problem?

Are real-time AI programs a mirage? Can an AI program ever be expected to always respond within a fixed latency, or must we start planning for only stochastic response latencies? What sorts of scheduling capabilities do AI programs require beyond those useful for other real-time programs?

Overall Goal of Workshop and Summary

Potential contractor/developers of defense software systems have little incentive to make investments in standards or tools for the uncertain likelihood of future contracts. Since AI capabilities are new, there is no established pool of experience in the defense software contracting industry which can fight for the language changes and tools which will make AI programming easier, cheaper and more effective. There are, on the other hand, major established groups to argue for better signal processing support, better decimal/mainframe support, better network support, better real-time support for traditional software control loops, etc. It is therefore likely that significant "holes" exist in the Ada language and infrastructure, which will only become evident later, when projects become late and costs balloon.

The overall goal of this workshop is a document which clearly states the requirements for programming languages to support real-time embedded AI programs for defense applications. These requirements need to be prioritized, and the consequences and costs of not meeting the requirements need to be estimated. Since modern warfare puts the ultimate premium on up-to-date intelligence, efficient resource allocation, and pin-point accuracy, AI will play a pivotal role in making sure that the weapons are located at the right place and the right time, and used against the right target with the appropriate ammunition with sufficient accuracy and concentration to knock out the target once, but only once. We have to make sure that we are fighting the *next* war rather than the *previous* war.

REFERENCES

Ada83. *Reference Manual for the Ada® Programming Language.* ANSI/MIL-STD-1815A-1983, U.S. Gov't Printing Office, Wash., DC, 1983.

Baker, Henry. "List Processing in Real Time on a Serial Computer". *Comm. of the ACM 21*,4 (April 1978),280-294.

Baker, H.G. "The Automatic Translation of Lisp Applications into Ada". *Proc. 8'th Conf. on Ada Tech.*, Atlanta, GA (March 1990),633-639.

Baker, H.G. "Structured Programming with Limited Private Types in Ada: Nesting is for the Soaring Eagles". *Ada Letters XI*,5 (July/Aug 1991), 79-90.

Baker, H.G. "Object-Oriented Programming in Ada83—Genericity Rehabilitated". *Ada Letters XI*, 9 (Nov/Dec 1991), 116-127.

Baker, H.G. "CONS Should rot CONS its Arguments, or A Lazy Alloc is a Smart Alloc". ACM *Sigplan Not. 27*,3 (March 1992), 24-34.

Baker, H.G. "The Treadmill: Real-Time Garbage Collection without Motion Sickness". ACM *Sigplan Not. 27*,3 (March 1992), 66-70.

Baker, H.G. "Equal Rights for Functional Objects or, The More Things Change, The More They Are the Same". ACM OOPS Messenger, 1992, to appear.

Baker, H.G. "Iterators: Signs of Weakness in Object-Oriented Languages". ACM *OOPS Messenger*, 1992, to appear.

Barnes, J.G.P. *Programming in Ada: Third Edition.* Addison-Wesley, Reading, MA, 1989,494p.

Hosch, Frederick A. "Generic Instantiations as Closures". *ACM Ada Letters 10*,1 (1990),122-130.

Kernighan, Brian W., and Ritchie, Dennis. *The C Programming Language.* Prentice-Hall, Englewood Cliffs, NJ, 1978.

Kownacki, Ron, and Taft, S. Tucker. "Portable and Efficient Dynamic Storage Management in Ada". *Proc. ACM SigAda Int'l Conf., Ada Letters*, Dec. 1987,190-198.

Mendal, Geoffrey O. "Storage Reclamation Models for Ada Programs". *Proc. ACM SigAda Int'l Conf., Ada Letters*, Dec. 1987,180-189.

Perez, E.P. "Simulating Inheritance with Ada". *ACM Ada Letters 8*,5 (1988),37-46.

Rosen, Steven M. "Controlling Dynamic Objects in Large Ada Systems". *ACM Ada Letters* 7,5 (1987),79-92.

Schwartz, Richard L., and Melliar-Smith, Peter M. "The Suitability of Ada for Artificial Intelligence Applications". Final Report, Contract #AAG29-79-C--0216, SRI Int'l., Menlo Park, CA, May 1980,48p.

Smith, D. Douglas. "ALEXI—A Case Study in Design Issues for Lisp Capabilities in Ada". *Wash. Ada Symp. 5* (June 1988),109-116.

Steele, Guy L. *Common Lisp, The Language; 2nd Ed.* Digital Press, Bedford, MA, 1990,1029p.

Taft, Tucker, *et al. [Ada-9X] DRAFT Mapping Document.* Ada-9X Proj. Rep., Feb. 1991.

Taft, Tucker, *et al. [Ada-9X] DRAFT Mapping Rationale Document.* Ada-9X Proj. Rep., Feb. 1991.

Yen, Mike. "Adapting an AI-Based Application from its Lisp Environment into a Real-Time Embedded System". *Proc. AIAA Comps. in Aerospace VII*, Monterey, CA, (Oct. 1989),1114-1122.

Yen, Mike. "Using a Dynamic Memory Management Package to Facilitate Building Lisp-like Data Structures in Ada". *Proc. AIDA-90*, Nov. 1990, 85-93.

Ada9X Issues for AI Systems

Why are AI people here discussing Ada9X now?

- We screwed up! AI people have not participated in Ada9X process due to hubris, ignorance or waivers.
- AI people have a lot to offer Ada9X — AI people have traditionally been "the pioneers with the arrows in their backs" on new hardware, software, languages, etc.
- AI usually shows the future of computers in 5-10 years — AI polished OO concepts, program develop. envs. (CASE), "dynamic" languages, etc.
- AI complaints re Ada in 1980 were remarkably prescient.
- Large # of embedded systems of 1990's are "intelligent" — mission planning, robotic weapons, intel. WS, etc.

Ada9X Issues for AI Systems

What makes AI systems so special/different?

- Very complex and/or non-homogeneous data structures — every object has different # and size of attributes
- Processing is highly heuristic — few science and/or engineering methods (e.g., vector algebra) to fall back on
- Programs are constantly changing — may have 1-1 correspondence between SW versions and missions!
- Much of the data is in the form of procedures rather than passive data structures
- Very tentative processing — try many approaches, take the best one so far
- Interacting "experts" communicating via a common "blackboard"

Ada9X Issues for AI Systems

- Can AI programs be successfully written and deployed in Ada?
- Does Ada9X make writing/deploying AI programs easier?
- Are there any "gotchas" in Ada9X w.r.t. AI applications?
- Can extremely large programs (AI or otherwise) be efficiently developed, delivered and maintained in Ada9X?
- Can the late-binding required in AI programs be achieved in Ada9X?
- Is Ada9X still too static for AI applications that may need to be reconfigured (recompiled??) for every mission?
- Does Ada9X support efficient access to complex shared data in the form of "blackboards" and/or "O-O databases"?

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies of Ada —

- Procedures as Values
- Garbage Collection
- Additional Tasking Control
- Consistent Definition of Parameter Binding

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Desirable but Non-Essential Capabilities for Revised Ada —

- True Abstract Data Type Facility
- Expanded Exception Handling
- Partial Parameterization

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Recommended Extensions for AI "Annex" —

- Procedures as Parameters
- Pragma Garbage_Collection
- More General Type Initialization
- Generic Types
- Type Instantiation Parameters

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

- crucial for "Late Binding"
- used as "Domain Experts"
- used to implement "Lazy Evaluation"
- used to emulate "infinite" data structures
- requires first-class "upward" (not LIFO) closures

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

Generics are NOT good enough!

- Generics can only accept CONSTANT procedural arguments
- Generics cannot take generic subprograms as parameters
- Generics cannot be renamed
- Generics are not inherited during derivation
- Generics CAN produce other generics—e.g., Currying

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

"Downward" Procedure Arguments would be Excellent

- Pass a subprogram as argument to subprogram call
- Subprogram argument has access to all "free variables"
- No storing of procedures in data — i.e.,
no possibility of subprogram closure argument "escaping"
- Can be used in a wide variety of situations requiring
complex control structures—e.g., "continuation-passing"
style and "call-backs".

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

First-Class "Upward" Procedure Values would be Better

- Subprogram closures can "escape" LIFO restrictions
- Have to manage storage for closures
- General solutions requires full garbage collection

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

Emulating Closures with C-style Functions w/o "Free Variables"

- User must design & manage own closure datatype
- Closure datatype needs union limit type to hold any Ada type
- Closure datatype includes pointer to subprogram which takes closure structure itself as an argument
- Closure datatype is a recursive datatype
- Can be done with a tool; otherwise, significant possibility of errors

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

Emulating Subprogram Arguments in Ada-83

- Use enumeration type to indicate which procedure to call
- Must build large "funcall" procedure which does case dispatch on enumeration type to correct procedure
- Must have separate "funcall" procedure for each subprogram type signature
- Maintenance nightmare
- Ada-9X MUST HAVE at least C-style function values

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Garbage Collection

- Ada is type-safe => no "dangling references" under normal conditions

- Ada-83 allows for full tracing GC, but only Ada-ED offers GC, and it is very slow (but not because of GC)

- GC normally pervades language implementation — very difficult/expensive to "add on" later

- Great difficulties in real-time and/or distributed environments — on the "bleeding edge" of research

- "Generational" GC's which are good for development envs. are not appropriate for embedded systems

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Garbage Collection

Emulating GC in/on-top-of Ada

- Just barely possible in safe, portable Ada-83
- Use limited private type which can manage all "roots"
- Only LP type can create/destroy a root, so LP has "birth control" for roots of the graph to be traced
- Requires extensive use of "in out" parameters
- Can't use traditional nested function composition notation
- Doesn't scale well to parallel threads and tasking

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Garbage Collection

Emulating GC in/on-top-of Ada — Problems

- Can't use traditional nested function composition notation
- Doesn't scale well to parallel threads and tasking
- Doesn't scale well to OO programming styles
- Ada83 won't allow programmer to disallow variable creation outside of defining package; have to "fake" it
- Ada83 generics can't be renamed to avoid "dotted" references
- Ada83 "in out" can't decide between "by reference" or "copy-in, copy-out"

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Garbage Collection

Are there any Alternatives?

- Explicit Free — dangerous, especially in large systems, where responsibility for deallocation becomes distributed

- Reference Count — inefficient (5X slower than GC); difficult to guarantee ref. count; interference probs. in parallel environment

- Linear (Unity Ref. Count) Structures — inefficient, with too much copying; difficult to guarantee unity reference count

- Don't Free, use infinite virtual memory — works in some non-persistent environments, but impractical for embedded sys.

- No "silver bullets" for storage management problems

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — More Tasking Control

- use tasks/parallel processing for evaluating multiple alternatives in parallel
- need "fair" timesharing to give all alternatives a chance to proceed
- requires ability to dynamically vary priorities based on "progress" towards a solution
- need to asynchronously & immediately kill useless processes
- need to be able to enumerate all of one's own subtasks to adjust priorities and/or terminate them
- need to detect and break deadlocks more aggressively

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Parameter Binding Modes

- Ada does not provide clean notion of "object identity"
- "in out" "mode" provides closest match (used in GC scheme)
- uncertainty about "by reference" or "copy-in/copy-out" is intolerable
- portable program MUST be inefficient, because it must assume the worst features of both implementations!
- "in out" variables MUST NOT BE ALIASED, yet compilers & run-time systems not required to check this
- recommend "in out" be copy-in/copy-in, add new "ref" mode

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Parameter Binding Modes

Interactions with Exceptions

- Uncertainty of mechanism means no consistency of state for exception handlers

- No PORTABLE way to write exception handlers

- separate "in out" and "ref" modes would clarify semantics and allow the programmer to optimize

- "ref" mode in non-shared memory implementation would be very inefficient, but programmer would know that

"The Suitability of Ada for Artificial Intelligence Applications"

Schwartz, R.L. & Melliar-Smith, P.M., SRI, May 1980

Critical Deficiencies — Procedures as Values

Technical Nits in Ada-83

- "Anonymous" subprogram syntax would be nice to eliminate the clutter of opening up a local declarative block just to define one subprogram which is used exactly once!

- The only recursive types in Ada-83 are pointer types; with procedure values, we now need recursive procedure types

APPENDIX G

MESSAGE FROM THE APPLICATIONS EXPERIENCES AND LESSONS LEARNED PANEL CHAIR

SIGAda Artificial Intelligence Working Group

Summer '92 SIGAda Workshop

Janet Faye Johns

The MITRE Corporation

Bedford, Massachusetts

617-271-8206

This paper assesses the current state-of-the-art for Artificial Intelligence (AI) applications development processes. The human and safety reasons why we need to "engineer" AI applications is discussed and information about the current processes used to develop knowledge-based systems is presented. Issues associated with requirements analysis, design methodologies, development techniques, test and validation, and maintainability are discussed for AI applications and for AI with Ada applications.

Viewgraphs are in this paper with an accompanying discussion section that is divided into two parts. The first states current issues while the second part provides a summary of the related workshop discussions.

APPLICATIONS EXPERIENCES AND LESSONS LEARNED PANEL

MESSAGE FROM THE PANEL CHAIR

BACKGROUND

During the past year, I have been collecting information about existing Artificial Intelligence (AI) applications developed with the Ada programming language for the SIGAda Artificial Intelligence Working Group's (AIWG) first annual survey [3]. This was a challenging task for a number of reasons, but I have come to believe that the major impediment to my data collection efforts is the state-of-the-art of developing AI applications. The basic nature of specifying and developing AI systems is a major stumbling block to formulating detailed software metrics and other measures that are so common for "engineered" systems. AI systems generally begin with few documented requirements and are developed with a series of evolutionary prototypes. Typical software metrics are not readily available for AI systems. This fact led me to investigate the current processes used to develop AI applications and the problems of applying current software engineering principles to AI applications.

This briefing was presented at the general session of the Summer '92 SIGAda conference. I presented this material as a devil's advocate challenge that stated the issues in an effort to generate open discussions about the issues. These and many other issues were discussed during the workshop. For the publication of this briefing in the workshop proceedings, I have added relevant information from the workshop discussions to the briefing. My briefing combined with my interpretation of the workshop discussions is the method I have chosen to document the workshop discussions for these proceedings.

REFERENCES

1. P. Collard and A. Goforth, November/December 1988, Knowledge Based Systems and Ada: An Overview of the Issues, Ada Letters, pp 72-81.

2. C. Culbert, D. Hamilton, and K. Kelley, 1991, State-of-the-Practice in Knowledge-Based System Verification and Validation, Expert Systems With Applications, Vol. 3, No. 4, pp 403-410.

3. J. Johns, June 1992, 1991 Annual Report for the ACM Special Interest Group for Ada Artificial Intelligence Working Group, MITRE Document M92B0000056.

4. N. Leveson, May 1992, High Pressure Steam Engines and Computer Software, Keynote address at the 14th International Conference on Software Engineering, pp 2-14 of the Proceedings.

5. Xiaofeng Li, September 1991, What's so bad about rule-based programming?, IEEE Software, Vol. 8, No. 5, pp 103-105. Editor's Note: Paul Sanders responded to this article in the January 1992 issue of IEEE Software.

6. D. Woods, April 1992, Space Station Freedom: Embedding AI, AI Expert, Vol. 7, No. 4, pp 32-39.

SIGAda Artificial Intelligence Working Group

Janet Faye Johns

22 June 1992

MITRE

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

AI with Ada: Where are we and where are we going?

Past:
Empirical evidence of past successes

Present:
Common issues face the AI and Ada communities

Future:
Our challenge is the application of sound software engineering principles when we write software to solve our problems.

JFJ 6/92

AI with Ada: Where are we and where are we going?

The 1991 AIWG industry survey [3] provided empirical evidence that the Ada programming language has been used successfully to develop Artificial Intelligence (AI) applications. The difficulty encountered in collecting software metrics for the 1991 AIWG survey raised many questions about the current state-of-the-art for AI requirements analysis, design, and development. We are not alone in our struggle to understand where we have been and where we are going. Recently, the "software crisis" and the emphasis on building trusted systems has caused many of us to examine the state of the entire computing industry.

For the present, many challenges face the AI community that are equally applicable to the AI with Ada community. These include domain specific technical issues; liability issues associated with building trusted systems; the difficulties with requirements analysis for AI systems; design methodologies which adequately encompass rapid prototyping; testing, validation, and verification techniques; maintainability for a long systems life cycle. The AI community and the AI with Ada community need to develop a cooperative effort that can focus our limited resources on developing solutions for our common problems.

Software has come a long way in the past few years and has surpassed hardware in development and maintenance costs. Sound software engineering principles is one proven solution used to control and manage the apparent out of control software development and maintenance costs. For the future, software engineering is a challenge that must be met by both the AI and the AI with Ada communities if we are to manage AI software development efforts and ensure that our systems are both supportable and maintainable for a long life cycle.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Software Safety and Liability Issues

"... The introduction of computers into the control of potentially dangerous devices has led to a growing awareness of the possible contribution of software to serious accidents.

...

Our greatest need now, in terms of future progress rather than short-term coping with current software engineering projects, is not for new languages or tools to implement our inventions but more in-depth understanding of whether our inventions are effective and why or why not.

..."

Nancy Leveson, May 1992, High Pressure Steam Engines and Computer Software *Keynote address at the 14th International* Conference on Software Engineering, p 2 - 14 of the Proceedings.

JFJ 6/92

Software Safety and Liability Issues

Our ability to "engineer" AI applications is rapidly becoming a deciding factor in the acceptance of safety-critical systems that contain AI applications. After several decades of successful research and prototyping, we are just beginning to grapple with the issues of building "trusted systems" that are safety critical, correct, dependable, robust, efficient, and secure. Other important aspects of building trusted systems that we must consider include understanding the human impact of our AI applications and the liability issues associated with AI applications. The credibility that software engineering provides for a software system will be major factor in our ability to gain support and public acceptance for our AI applications.

How well do we really understand our technology? One of the questions that Nancy asked in her keynote address is: "When software fails, why can't it fail in a safe state?".

You may argue that software can't be unsafe in the sense that software does not physically harm people; but, software now controls hardware that can physically harm people. In our technologically dependent society, embedded software surrounds us in our home, at the office, in our automobiles, and in airplanes. The average 1992 automobile is now estimated to contain more software than NASA used to accomplish the 1967 lunar landing mission. Software in modern aircraft is measured in millions of lines of source code. These facts should be sufficient for all of us to be concerned about software safety.

Articles in magazines and trade journals have begun to discuss legal liability issues such as who is liable and responsible when an AI system provides bad advice or erroneous data.

How many of us could prove that our software will always fail in a safe state?

Are our validation and verification techniques sufficient to prove our software will either never fail or will always fail in a safe state?

How well do we understand the software engineering processes that we use to develop software so that our software is as safe as possible?

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

We must understand the human impact of our inventions.

"... A major airline, known for having the best aircraft maintenance program in the world, a few years ago introduced an expert system to aid their maintenance staff. The quality of maintenance fell. The staff began to depend on the computerized decision making and stopped taking responsibility and making their own decisions. When the software was changed to provide only information and only when requested, quality again rose.

...

The use of computers to enhance safety may actually achieve the opposite effect if the environment in which the computer will be used and the human factors are not carefully considered.

..."

Nancy Leveson, May 1992, High Pressure Steam Engines and Computer Software Keynote address at the 14th International Conference on Software Engineering, p 2 - 14 of the Proceedings.

JFJ 6/92

We must understand the human impact of our inventions.

A common bond among all computer scientists is that we must strive to understand the human impact of our systems. For artificial intelligence (AI) systems that interface with people, we must understand how our systems are actually used. This factor may mean the difference between success or failure for our systems.

In Nancy's example, the underlying "intelligent" system was ineffective when the system made decisions; but, was very effective at providing only solicited advice in response to human requests. This example stresses the importance of the human element in our systems.

How many of us consider these human factors in the design of our AI systems?

How do you specify requirements for these human factors?

How do you measure the correctness of the human factors requirements?

Have we even begun to consider *how a human being interacts with* and reacts to an AI system that is constantly learning and growing thereby producing unpredictable behavior?

Workshop Discussions

This topic received little attention during the workshop discussions. However, several participants did say that more attention must be paid to the human impact of our AI systems.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Theorem: $\exists$ An optimal software engineering process

$\forall$ Artificial Intelligence Problems

Proof:

Ada promotes good software engineering

...

Ada has been used to solve real world AI problems

...

$\therefore$ Ada provides an optimal software engineering process for all Artificial Intelligence problems.

Theorem: An optimal software engineering process exists for all Artificial Intelligence problems

Our greatest challenge is the application of software engineering principles to the development of AI applications. This theorem and proof are my way of asking the question:

"Are we using sound software engineering principles to develop solutions for our Artificial Intelligence (AI) problems?"

Ada promotes good software engineering practices; but, does Ada support an optimal software engineering process for all AI problems? If not all AI problems, does Ada support an optimal software engineering process for some selected subset of AI problems? What are the issues associated with using Ada to develop AI applications?

Workshop Discussions

An assessment of the experiences of using Ada to develop AI applications shows that many of the issues are equally applicable to all AI development environments. Unfortunately, these issues represent software engineering challenges that span the complete software life cycle:

1) Requirements are difficult, if not impossible, to define until the AI application has been prototyped and developed.

2) Requirements analysis requires rapid prototyping.

3) Design can only be accomplished with iterative prototypes.

4) Test, verification, and validation are difficult and ill-defined for AI software.

5) Little maintenance experience is available for AI applications because few large-scale systems have been fielded.

6) Based on the developer's experiences with prototype knowledge maintenance, knowledge maintenance for a long life cycle is an area of critical concern .

7) Scalability from the requirements analysis and design prototypes to a full scale system is an ill-defined area for AI software.

SIGAda Artificial Intelligence Working Group Summer '92 SIGAda Workshop

Our Challenge: Software Engineering with Ada and AI

"One of the most complex systems ever taken into a hostile environment already has projects that leverage AI technology.

...

Ada has been chosen as the Freedom Station language of choice, primarily because of its software engineering characteristics: abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability.

...

This "design for evolution" is accomplished by conforming to standards (UNIX, FDDI, Ada, X Windows) and designing the interconnections with sufficient bandwidth to allow for faster components as they become qualified for space.

..."

Donald Woods, April 1992, AI Expert, Vol 7, No 4, pp 32-39, Space Station FREEDOM: Embedding AI

JFJ 6/92

Our Challenge: Software Engineering with Ada and AI

The future is bright for both AI and Ada. What does the future hold for AI with Ada? The answer to this question will depend on our ability to use sound software engineering processes in the development of AI systems with Ada -- or any other language.

Workshop Discussions

The 1991 AIWG annual applications survey [3] documented 17 AI with Ada products, eight (47%) of which were less than 5 thousand source lines of code (KSLOC). The AI with Ada applications described in these proceedings and at the workshop represent a sampling of much larger AI with Ada applications with sizes of 40, 45, 70, 153, and 220 KSLOC. Obviously, the future is very bright for AI with Ada applications.

The 1991 AIWG annual applications survey identified the need for software metrics which are virtually nonexistent for AI applications. Participants at the workshop agreed that software metrics are difficult for AI systems and were not surprised about the absence of metrics for AI with Ada applications. Unfortunately, this situation makes it very difficult to manage and cost AI projects.

There were several discussions about projected AI applications for large-scale systems such as the Strategic Defense Initiative (SDI). For large mission-critical systems such as SDI, proven software engineering processes must be developed and matured so that these large AI applications can be managed and accurately budgeted. All of the AI software engineering issues with requirements specification, iterative proof-of-concept design, test and validation, and maintainability will be very critical issues for large-scale embedded SDI applications.

The workshop participants described AI with Ada applications in various phases of the software life cycle. AI with Ada developers are continuing to design and develop applications. However at this point, there is very little experience with operations and maintenance for large-scale AI with Ada systems.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Can we bridge the gap between the Ada and AI cultures?

"... Cross-cultural vantage point. ... On one side is the Ada community that represents the engineer's "how to" culture; and, on the other side, the AI community that represents the scientist's "what" and "why" culture.

...

A fundamental thesis of AI is that "intelligence" can be reasonably approximated by computation; i.e. a computer program.

..."

Phillippe Collard and Andre Goforth, November/December 1988, Knowledge Based Systems and Ada: An Overview of the Issues, Ada Letters, p 72 - 81.

JFJ 6/92

Can we bridge the gap between the AI and Ada cultures?

The representation of knowledge, manipulation of knowledge, reasoning with knowledge, understanding the true meaning of intelligence, and the impact of our systems on intelligent beings are some of the exciting challenges facing all Artificial Intelligence (AI) researchers today. These challenges form a common bond between the AI with Ada community and other AI researchers.

The gap that separates the AI and Ada communities includes many issues associated with our basic problem solving approaches; that is, rapid prototyping versus a software engineering process that begins with a set of well-defined requirements.

Can we develop software engineering processes that encompass the "what" and "why" AI culture?

Workshop Discussions

Rapid prototyping techniques are being used with the Ada programming language. Based on the discussions at the workshop and the AI applications described in these proceedings, the basic AI problem solving approach of rapid prototyping is being used to implement AI applications with the Ada programming language.

The general consensus of the workshop participants is that their AI with Ada projects begin with few well-defined requirements. Therefore, the lack of well-defined requirements is a common characteristic for both AI and AI with Ada applications.

Workshop participants expressed the need for a rich set of AI with Ada tools like those typically found in other AI environments. Several discussions centered around the need for requirements analysis, design, development, maintenance, and real-time analysis tools. These tools could support an environment that approaches the "what" and "why" AI culture while enforcing proven software engineering processes. More research is definitely required to develop tools that support a rich AI software engineering environment.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Requirements Definition with Prototypes

"... Much work is needed to include the concept of rapid prototyping in the software engineering process. Ada's structure is based on the assumption that the design of a software system is derived from a set of crisp requirements. AI applications are not defined so much by crisp requirements but rather by the available knowledge on a particular problem domain that can be integrated in a computer program. To successfully implement AI applications in Ada, one will have to merge these two approaches.
..."

Phillippe Collard and Andre Goforth, November/December 1988, Knowledge Based Systems and Ada: An Overview of the Issues, Ada Letters, p 72 - 81.

Requirements Definition with Prototypes

This article appeared four years ago in the November/December 1988 issue of Ada Letters. How successful have we been with the merger of rapid prototyping with the Ada software engineering process?

Workshop Discussions

Based on the workshop discussions and the AI applications described in these proceedings, rapid prototyping is being used for AI with Ada applications. Although textbook solutions have not been developed, progress has been made in the use of rapid prototyping for AI with Ada applications. All of the AI with Ada applications described in these proceedings used prototyping which provides empirical evidence that prototyping with Ada is not only technically feasible but has become a normal part of the process of developing AI with Ada applications.

The issue of design freedom was briefly discussed at the workshop. Government participants described the situation where they develop prototypes to define requirements and as a consequence develop an implementable design that isn't used by the development contractors because the contractor has design freedom under government acquisition regulations. With the iterative "learning by doing" nature of AI applications, the development contractor frequently repeats the same iterative cycle to design the AI application. Mechanisms to transfer prototype design information to the full-scale engineering development phase will continue to be a topic of discussion.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Maintainability is a big factor in the software life cycle.

"...
If AI and Ada are to coexist, an important element will be the ability for software engineers to understand knowledge engineering problems, and for AI specialists to take into account the requirements imposed by large, long lived projects.
..."

Phillippe Collard and Andre Goforth, November/December 1988, <u>Knowledge Based Systems and Ada: An Overview of the Issues.</u> Ada Letters, p 72 - 81.

JFJ 6/92

Maintainability is a big factor in the software life cycle.

Two-thirds of the life cycle cost a software system is currently being attributed to maintenance and two-thirds of this cost is spent on product improvements and enhancements.

Knowledge and the way we interpret knowledge are very dynamic elements in Artificial Intelligence (AI) systems. Our understanding of knowledge, and how to use it, is constantly changing. For a typical AI system, knowledge -- the knowledge base, the fact base, the inference engine, reasoning model, etc. -- must be maintainable for a long life cycle.

One of the promises of the Ada programming language is that software will be maintainable for a long life cycle. Large, long-lived software projects require that the software be maintainable, supportable, and reliable. This in turn demands a software engineering approach that includes a thorough requirements analysis, a flexible design, and extensive test activities.

How well does Ada support the ever-changing nature of a knowledge based system?

Workshop Discussions

In general, large-scale AI applications have not been operational long enough for much maintenance experience. Software implementation techniques impact maintainability and performance. The AI applications described in these proceedings include three approaches to implementing expert systems:

1) code the rules directly in Ada for runtime performance

2) use two modes for the rule base: an interpretive mode for rule execution during development and a runtime mode that involves generating Ada code for the rules to improve runtime performance

3) a runtime mixture of interpreted rules and rules implemented in Ada.

One of the developers who is using the first approach of implementing the rules in Ada lamented about the tremendous overhead -- approximately 1 week -- of recompilation for any changes to the 510 rules in his rule base.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Are AI with Ada systems testable, maintainable, and reliable?

"...
From a software engineering perspective, rule based programming is a failure because systems developed in it are unmaintainable, untestable, and unreliable.

...
One of the best-known advantages of rule-based programming is that it minimizes the distinction between development and maintenance, because the developer and maintainer deal only with production rules.

...
Maintenance could eventually consume up to 60 percent of the total cost of a product in its entire life cycle. Unfortunately, this advantage is not realized in practice. ..."

Xiaofeng Li, September 1991, What's so bad about rule-based programming?, IEEE Software, Vol 8, No 5, p 103 - 105. Paul Sanders responded to this article in the January 1992 issue of IEEE Software.

JFJ 6/92

Are AI with Ada systems testable, maintainable, and reliable?

The difficulties inherent in AI requirements analysis inevitably lead to problems with the testing, verification, and validation (V&V) of AI applications. Testing and validation activities ensure that the developed software system satisfies a well-defined set of requirements which unfortunately do not normally exist for AI applications. Verification activities ensure that the developed system is supportaole and maintainable which raises issues associated with the feasibility of verifying non-deterministic systems that can learn and adapt over time. Testing and V&V are critical areas of current research because the public and the software engineering community have begun to focus attention on building trusted systems that are correct, dependable, robust, safety critical, efficient, and secure.

Workshop Discussions

The panel discussions regarding testing and V&V led to two interesting thoughts. First, AI applications seek to emulate human intelligence and behavior. How do we test and validate humans? In general, the proof of human intelligence and their ability to learn is through on-the-job performance. Therefore, in essence, humans do not undergo the same scrutiny of test and V&V that we are trying to impose on AI applications. Second, instead of trying to perform an unnatural test and V&V scrutiny of AI applications, perhaps it would be more meaningful to develop techniques and tools that determine if an AI application is fit-for-purpose; that is, does it match the problem domain, is it operable in the proposed environments, and can it be learned with relative ease.

Workshop participants identified a critical need for tools that support the full life cycle of AI applications. A real-time browser that would enable developers and maintainers to "peek" into the system was identified as a critical tool for AI applications. In general, the larger the application, the more valuable these tools would become. Problems identified during the tool discussions covered issues such as version walk between the operational system and tools. One suggestion was to develop the tools in Ada and maintain the tools with the AI application. This line of reasoning led to the conclusion that we should be developing a set of useful AI tools and that the AIWG should maintain a database of available tools.

SIGAda Artificial Intelligence Working Group Summer '92 SIGAda Workshop

Case Study: Accuracy of Existing Systems

From a survey of 65 AI developers and 5 users:

75% of the systems performed diagnosis
73% of these were in the aerospace industry

The systems were less accurate than expected:

75% were less accurate than expected
  39% were less than 90% accurate
  54% were less than 95% accurate

62% were less accurate than an expert

David Hamilton, Keith Kelly, and Chris Culbert, 1991, <u>State-of-the-Practice in Knowledge Based System Verification and Validation,</u> Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

JFJ 6/92

Case Study: Accuracy of Existing Systems

Culbert, Hamilton, and Kelly documented the experiences and problems encountered by knowledge based systems developers and users. In their paper, the terms knowledge based systems and expert systems were used interchangeably by the authors. Although the focus of the authors was verification and validation, the article reveals some very valuable information about the state-of-the-art of knowledge based systems. The article described responses from 65 developers and 5 users. The surveys were distributed to a selected set of users and developers as well as at conferences so that there were a wide variety of survey participants. The authors plan to use the problems uncovered in the survey to make recommendations that will improve the quality of knowledge based systems used for the Space Station Freedom Project.

75% of the surveyed systems performed diagnosis and 73% of these diagnostic systems were used in the aerospace industry. The users and developers in this survey felt that their systems were less accurate than their expectations. Further, 62% were less accurate than a human expert. These facts stress the urgency of addressing software safety and liability issues for AI applications.

As most AI systems are developed without well-defined requirements, how do you quantify accuracy expectations for AI applications?

Workshop Discussions

Accuracy workshop discussions focused on the performance constraints of real-time systems. Basically, an AI system could provide the best possible answer within its timing constraints. After all, this is what humans do, humans provide the best possible answer in the allotted time.

There were no discussions about quantifying accuracy expectations for AI applications. This is obviously an area for future research.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Case Study: Development Without Requirements

52% of the systems had no documented requirements

"...In some cases, requirements were not written because it was felt that a requirements document was a formally written paper that needed to be approved before development could proceed. In other cases, an iterative prototyping development effort took place and was followed by documenting system requirements.
...
There is little understanding of how requirements for an expert system should be generated and documented.
..."

David Hamilton, Keith Kelly, and Chris Culbert, 1991,
<u>State-of-the-Practice in Knowledge Based System Verification and Validation,</u>
Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

JFJ 6/92

Case Study: Development Without Requirements

52% of the 70 AI systems in the survey had no documented requirements! For those working in the DoD-STD-2167A environment or another modern software engineering environment, this is a frightening statistic because these environments rely on a set of well-defined requirements to manage, scope, and budget software projects. How effective are the processes used to develop AI systems without a set of well-defined requirements? What is even more difficult to understand is how you design, develop, and test AI systems without requirements.

We normally view an Ada development in terms of a set of well-defined requirements. This statistic represents a challenge to the implementation of AI systems with Ada. We know that Ada is being used to develop AI systems. Are Ada developers using the same techniques that are being used in other AI environments and languages?

Workshop Discussions

Based on the information presented at the AIWG workshop, Ada developers are developing AI systems with few well-documented requirements. Ada developers are also using iterative, rapid prototyping techniques similar to those used in other AI environments. The projects described *in these proceedings* used prototyping techniques to perform requirements analysis as well as to iteratively design and develop AI with Ada applications.

Requirements for AI applications are defined through iteration, that is learning by doing. AI requirements are difficult, if not impossible, to specify without prototypes. Most of the effort for AI projects is spent defining requirements and prototyping. In the case of the 70,000 source line of code (SLOC) Ada application described in these proceedings, 2/3 of the calendar time and labor were expended developing prototypes to define the requirements for the AI application.

AI requirements analysis requires a flexible process because defining AI requirements is an evolutionary process. This type of process is not a fixed price problem and the current DoD-STD-2167A may be too rigid for AI applications even with the software engineering discipline provided by the Ada programming language. We discussed many AI specific issues associated with the software engineering process and DoD-STD-2167A. These discussions led us to question whether AI requirements and design are the same as understood in the DoD-STD-2167A and other software engineering environments. Based on these discussions, the AIWG decided to be more active in expressing AI specific requirements and concerns to the standards bodies that are developing standards which impact the development of AI applications with the Ada programming language.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Case Study: Design Methodologies

"...
40% used a cyclic model for development. However,
22 % of the respondents stated that no model was followed.
..."

```
Loop    -- cyclic development model
           define requirements
           design
           rule generation
           prototype
End loop
```

David Hamilton, Keith Kelly, and Chris Culbert, 1991,
State-of-the-Practice in Knowledge Based System Verification and Validation,
Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

JFJ 6/92

Case Study: Design Methodologies

Many of the survey respondents did not use a formal design and development methodology. What are the design methodologies used to develop AI with Ada applications?

How well do these design methodologies support a long system life cycle for a constantly changing knowledge based system?

How well do the Ada software engineering characteristics of abstraction, information hiding, confirmability, etc. support an ever-changing knowledge based system?

Workshop Discussions

Based on the workshop discussions, AI with Ada developers are using a variety of design techniques and all of these techniques involve prototyping.

Workshop participants identified a critical need for tools that support the complete software life cycle in addition to an iterative design process. There was general agreement that the evolutionary nature of AI applications requires prototyping and a spiral or cyclic design process.

There were several discussions about the use of fourth generation languages (4GLs), language translators, and code generators. Should 4GLs or language translators be allowed in an Ada development? If translators are allowed, what is maintained -- the source language or the generated Ada? If the maintainer does not have the same tools as the developers then all maintenance advantages will be lost. How do you ensure that the language translation scenario will be usable and supportable for a long life cycle? There was general agreement that translation and code generation tools could be written in Ada and maintained as part of the Ada software to ensure that the tools had the same life cycle as the generated software.

SIGAda Artificial Intelligence Working Group Summer '92 SIGAda Workshop

Case Study: Operational Prototypes

"...
Although we attempted to get respondents to state that their system was either 'a prototype' or 'operational', we received indications that the distinction was often difficult to make.
..."

David Hamilton, Keith Kelly, and Chris Culbert, 1991,
State-of-the-Practice in Knowledge Based System Verification and Validation,
Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

Case Study: Operational Prototypes

As discussed in the survey, 70% of the systems were operational and the rest were considered prototypes but some of these prototypes had operational users. The respondents could not clearly distinguish between prototypes and operational systems -- their systems were operational prototypes. How does this "reality" fit with the Ada design and development methodologies?

What are the software engineering implications of this statistic with respect to:

-- requirements analysis
-- design and development
-- test methods
-- verification and validation
-- configuration management
-- operations and maintenance?

Workshop Discussions

Will your prototype ever grow up and leave home? Prototypes tend to live long beyond the AI requirements analysis and proof-of-concept design phases of the systems life cycle. Although these proceedings document prototypes that have been successfully transitioned into operational systems, workshop discussions reveal that more research is necessary to develop processes that support this transition -- assuming that this transition is desirable. In some of the workshop discussions where participants described their operational prototypes it was apparent that the developers have maintained a very close relationship with the operational users and the developers were still responsible for the prototypes.

Prototypes and human understanding of the problem domain are the standard legacy of an AI requirements analysis effort. What happens to this legacy? We would like to use our human understanding to specify the desired system in clear unambiguous requirements, but this is rarely possible. Prototypes typically reflect our understanding of how to implement a solution and in many cases are required to prove that it is possible to "engineer" the full scale system. Can you successfully "re-engineer" a prototype into a supportable and maintainable system? Based on panel experiences described in these proceedings, Ada prototypes are being re-engineered into supportable and maintainable systems.