P-58

A Distributed Agent Architecture for Real-Time Knowledge-Based Systems

Final Report

Real-Time Expert Systems Project - Phase I

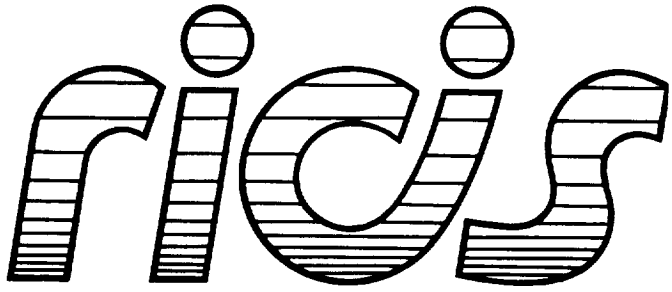
S. Daniel Lee

Inference Corporation

May 1990

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

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



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

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

A Distributed Agent Architecture for Real-Time Knowledge-Based Systems

Final Report

Real-Time Expert Systems Project - Phase I

Preface

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

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

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

A Distributed Agent Architecture for Real-Time Knowledge-Based Systems

Final Report, Real-Time Expert Systems Project - Phase I
prepared for
Research Institute for Computing and Information System (RICIS)
University of Houston - Clear Lake (UHCL)
and
NASA Johnson Space Center (JSC)
under
Subcontract 015
RICIS Research Activity SE.19
NASA Cooperative Agreement NCC-9-16

S. Daniel Lee

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

May 1990

Table of Contents

1. Introduction 1
2. Advanced Automation Projects for NASA's Space Station Freedom 3
2.1 Current Projects 3
2.1.1 Operations Management System 3
2.1.2 Electrical Power System 4
2.1.3 Thermal Control System 5
2.1.4 Environmental Control and Life Support System 6
2.1.5 Summary 6
2.2 Requirements for Future Projects 6
3. Intelligent Real-Time Problem Solving Architectures 8
3.1 Current IRTPS Architectures 8
3.1.1 DVMT 8
3.1.2 Guardian 9
3.1.3 Phoenix 9
3.1.4 PRS 10
3.1.5 RT-1 10
3.1.6 SOAR 11
3.1.7 Summary 11
3.2 Problems in Current Architectures 12
4. Distributed Agent Architecture 13
4.1 Technical Objectives 13
4.2 Design Overview 13
4.3 Reactive Agents 14
4.4 Cognitive Agents 16
4.5 A Meta-Level Agent 17
4.6 Interagent Communication 18
4.7 APEX Testbed 19
5. Future Work 22
6. Conclusion 23
References 24
I. Ada-Based Expert System Tools 32
I.1 Commercial Tools 32
I.1.1 ART-Ada 32
I.1.2 CHRONOS 34
I.1.3 Classic-Ada 34
I.1.4 CLIPS/Ada 34
I.1.5 Summary 34
I.2 Research Tools 35
I.2.1 ABLE 35
I.2.2 ARTIE 35
I.2.3 ERS 36

I.2.4 FLAC 36
I.2.5 KEE/Ada 36
I.2.6 MeriTool 37
I.2.7 Other Related Work 37
II. A Proposal for Real-Time Extensions to ART-Ada 38
II.1 Introduction 38
II.2 Performance Monitoring and Tuning 38
II.3 Temporal Reasoning and Trend Analysis 39
II.3.1 Monitors 39
II.3.2 Events 40
II.3.3 Timers 41
II.4 Dynamic Rule Priority 42
II.5 Message Passing between Distributed Expert Systems 42
II.6 Other Issues 43
III. Feedback from Review Meetings 44
III.1 Introduction 44
III.2 Meeting with the Communications and Tracking Group 44
III.3 Meeting with MITRE 45
III.4 Meeting with Ford Aerospace and IBM 46
III.5 Meeting with McDonnell Douglas 47
III.6 Conclusion 47

List of Figures

Figure 4-1: Distributed Agent Architecture for real-time knowledge-based systems 14
Figure 4-2: Current APEX 19
Figure 4-3: APEX based on Distributed Agent Architecture 20
Figure I-1: Overview of ART-Ada 33

List of Tables

Table 2-1: Selected Advanced Automation Projects for SSF 6
Table 3-1: Selected IRTPS Architectures 11
Table I-1: Commercial Ada Tools 35

1. Introduction

The current, ongoing work of Inference, the "Real-Time Expert Systems" project for NASA Johnson Space Center, under a subcontract to the University of Houston - Clear Lake, has provided valuable insights into requirements for real-time knowledge-based systems being developed for NASA's Space Station Freedom. NASA's Space Station Freedom is an example of complex systems that require both traditional and AI real-time methodologies. The standard on-board processor on the Station is an 80836-based workstation with limited memory. In the ground-based control center, on the other hand, conventional engineering workstations can be used for AI applications. It has also been mandated that Ada should be used for all new software development projects.

The Station also requires distributed processing. For example, if expert systems for fault detection isolation and recovery (FDIR) for the Station were fielded only in the ground-based control center, communication delays could cause serious problems. Catastrophic failures on the Station can cause the transmission system to malfunction for a long period of time, during which ground-based expert systems cannot provide any assistance to the crisis situation on the Station. This is even more critical for other NASA projects that would have longer transmission delays (e.g. the Lunar base, Mars missions, etc.)

However, current real-time knowledge-based system architectures suffer from a variety of shortcomings:

- A heavy dependence on inefficient implementation platforms, usually Common Lisp, which makes it difficult if not impossible to be deployed in real-time embedded systems.
- A weak integration with traditional real-time computing methodologies.
- An inability for the architectures to be distributed among multiple heterogeneous platforms that communicate asynchronously.

We have, previously, implemented an Ada-based expert system tool, ART-Ada, to facilitate the deployment of expert systems in Ada, which addresses the first point above [Lee & Allen 89], [Lee & Allen 90], [Lee 90].

We propose a distributed agent architecture (DAA) that can support a variety of paradigms based on both traditional real-time computing and artificial intelligence. DAA consists of distributed agents that are classified into two categories: reactive and cognitive. Reactive agents can be implemented directly in Ada to meet hard real-time requirements and to be deployed on on-board embedded processors. A traditional real-time computing methodology under consideration is the rate monotonic theory that can

guarantee schedulability based on analytical methods [Sha 89], [Sha & Goodenough 90]. AI techniques under consideration for reactive agents are approximate or "anytime" reasoning that can be implemented using Bayesian belief networks as in Guardian [Hayes-Roth et al 89], [Hayes-Roth 89]. Fuzzy logic [Lim & Takefuji 90], [Zadeh 88], [Togai & Watanabe 86] and reactive planning [Agre & Chapman 87], [Drummond 88], [Kaelbling 88], [Nilsson 89], [Schoppers 87] are also being considered for reactive agents.

Cognitive agents are traditional expert systems that can be implemented in ART-Ada to meet soft real-time requirements. During the initial design of cognitive agents, it is critical to consider the migration path that would allow initial deployment on ground-based workstations with eventual deployment on on-board processors. ART-Ada technology enables this migration while Lisp-based technologies make it difficult if not impossible.

In addition to reactive and cognitive agents, a meta-level agent would be needed to coordinate multiple agents and to provide meta-level control. An important area of coordination is timeline management. Following [Ash & Hayes-Roth 90], we intend to implement three timelines --- occurred, expected, and intended --- where each timeline records one type of information. Any agents can process or post *events* in any timelines through the meta-level agent.

The proposed testbed for DAA is APEX (Autonomous Power EXpert), which is a real-time monitoring and diagnosis expert system for the electrical power distribution system of NASA's Space Station Freedom [Truong et al 89], [Walters et al 90]. APEX was previously implemented in KEE and Common Lisp and is being ported to ART-Ada and Ada at NASA Lewis Research Center. The main purpose of the APEX testbed is to demonstrate the advantages of DAA.

2. Advanced Automation Projects for NASA's Space Station Freedom

Advanced automation projects for NASA's Space Station Freedom have been initiated due to special interests from Congress [Dominick et al 89]:

> Congress has displayed substantial interest in accelerating the dissemination of advanced automation technology to and in U.S. industry. Space Station was selected as the high-technology program to serve as a highly visible demonstration of advanced automation, and spur dissemination of the technology to the private sector.

2.1 Current Projects

In this section, selected advanced automation projects for the Space Station Freedom are summarized, mainly from the proceedings of the third annual workshop on Space Operations Automation and Robotics (SOAR '89) and [Bayer 89].

2.1.1 Operations Management System

[Taylor & Hansen 89] describes OMS as follows:

> The Space Station Freedom Program (SSFP) Operations Management System (OMS) is a set of functions which includes application software and manual interactions with the Freedom Station either from onboard or on the ground. OMS requirements have been organized into five task groups: 1) Planning, Execution and Replanning, 2) Data Gathering, Preprocessing and Storage, 3) Testing and Training, 4) Resource Management, and 5) Caution & Warning and Fault Management for onboard subsystems.

A prototype FDIR (fault detection isolation and recovery) system for OMS is being developed in two phases: the development of an onboard communications network FDIR system and global FDIR for onboard systems [Taylor & Hansen 89]. OPS83 and C++ are being used for the prototype. The user interface of the prototype is being constructed using TAE+.

Another prototype implements a proof-of-concept for a subset of the onboard OMS software, the Operations Management Application (OMA) [Kelly et al 89]. This prototype was ported from a Lisp and Symbolics environment to a VAX environment. The OMA prototype has two components, the Procedures Interpreter (PI) and the Integrated Status Assessment (ISA).

> The PI demonstrates the role automation plays in intelligent commanding and monitoring of space station system operations as well as interactive support functions for the crew. The PI accepts procedures (formal, English-like

definitions for crew activities) as input. This procedure is provided to an engine that carries out the formulation and issuance of commands to be sent to the systems being affected. The PI receives and monitors status and configuration from the simulations. With this information, the PI maintains a "context monitor", to track the progress of the procedure. Concurrently, flight rules are enforced as required by the current operation. The PI takes action when an activity violates flight rules in the context of the current operation.

The ISA prototype is a rule- and model-based expert system that demonstrates Space Station Freedom fault detection and isolation. The ISA consists of a knowledge base, an inference engine, and a user interface. The knowledge base encompasses facts and rules. The facts contain a high level qualitative model of Space Station Freedom systems. The rules consists of generic fault isolation knowledge and system specific knowledge to determine the source of faults.

Originally, the ISA prototype was written in Lisp and OPS5. It was ported to C and CLIPS/C later. MITRE plans to port ISA to Ada using ART-Ada. The PI was ported from Lisp to Ada using the Operations and Science Instrument Support (OASIS) software, an Ada package written at the University of Colorado at Boulder [Jouchoux et al 87]. The ISA and the PI are integrated using VMS interprocess communication services.

2.1.2 Electrical Power System

APEX [Truong et al 89], [Walters et al 90] and SSM/PMAD [Walls & Lollar 89] are being developed for the electrical power system of the Space Station. [Walters et al 90] describes APEX as follows:

The Autonomous Power Expert (APEX) system has been designed to monitor and diagnose fault conditions that occur within the Space Station Freedom Electrical Power System (SSF/EPS) Testbed. The APEX system is being developed at the NASA Lewis Research Center.... APEX is designed to interface with SSF/EPS testbed power management controllers to provide enhanced autonomous operation and control capability.

The APEX architecture consists of three components: (1) a rule-based expert system, (2) a testbed data acquisition interface, and (3) a power scheduler interface. Fault detection, fault isolation, justification of probable causes, recommended actions, and incipient fault analysis are the main functions of the expert system component. The data acquisition component requests and receives pertinent parametric values from the EPS testbed and asserts the values into a knowledge base. Power load profile information is obtained from a remote scheduler through the power scheduler interface component.

APEX was originally written in KEE and Lisp and being ported to ART-Ada and Ada at NASA Lewis Research Center.

[Walls & Lollar 89] describes SSM/PMAD as follows:

> The Space Station Module Power Management and Distribution (SSM/PMAD) Breadboard, located at NASA's Marshall Space Flight Center (MSFC) in Huntsville, Alabama, models the power distribution within a Space Station Freedom Habitation or Laboratory module.... Three cooperative Artificial Intelligence (AI) systems manage load prioritization, load scheduling, load shedding, and fault recovery and management.... FRAMES (Fault Recovery And Management Expert System) monitors the system for anomalies. Maestro is a resource scheduler which can create a schedule based on multiple constraints. The LPLMS (Load Prioritization List Management Scheduler) keeps up with the dynamic priorities of all payloads and develops load shedding lists for contingencies which require load shedding.

FRAMES was implemented in Common Lisp and the Common Lisp Object System (CLOS) on a Xerox 1186 workstation. Maestro and LPLMS were written in Common Lisp on a Symbolics Lisp workstation.

2.1.3 Thermal Control System

[Dominick et al 89] describes TEXSYS as follows:

> The NASA Systems Autonomy Demonstration Project (SADP) was initiated in response to the above stated Congressional interest for Space Station automation technology demonstration. The SADP is a joint cooperative effort between Ames Research Center (ARC) and Johnson Space Center (JSC) to demonstrate advanced automation technology feasibility using the Space Station Freedom Thermal Control System (TCS) test bed.... In response to the TCS challenge, the project's expert system technology development has been concentrated in the following areas: (1) integration of knowledge-based systems into a complex real-time environment; (2) causal modeling of complex components and elements through representation of first principles, quantitative models, and qualitative models in the knowledge-base; (3) use of combined model-based and rule-based reasoning; and (4) use of trend analysis heuristic rules....
>
> This research has lead to the development and use of a multi-purpose Model Toolkit (MTK) and Executive Toolkit (XTK) for model-based expert systems. These tools were used to create TEXSYS (Thermal EXpert SYStem) ... that performs actual control of a system as well as conducting monitoring and fault diagnosis.

TEXSYS is developed on top of MTK and XTK, which are written in KEE and Common Lisp. [Bayer 89] further describes MTK as follows:

> MTK is a utility package developed for use with the KEE expert system development tool to provide support for model-based reasoning... MTK was originally inspired by the tool SIMKIT and includes capabilities similar to that package, but with additional extensions and enhancements. MTK is built on top of KEE version 3.1 and is closely integrated with a number of KEE utilities that provide basic support for a number of MTK functions.

2.1.4 Environmental Control and Life Support System

[Dewberry 89] discusses the advanced automation project for Environmental Control and Life Support System (ECLSS). In particular, a diagnostic prototype of the Potable Water Subsystem is described in [Lukefahr et al 89]. This system is implemented in CLIPS/C and Hypercard on an Apple Macintosh computer, and will be ported to ART-Ada.

2.1.5 Summary

Selected advanced automation projects for the Space Station Freedom are summarized in table 2-1.

System	Organ.	SSF Module	Initial Prototype	Porting
ISA	MITRE	OMS	OPS5/Lisp & CLIPS/C	ART-Ada
FDIR	Ford	OMS	OPS83/C++	?
APEX	LeRC	Electrical Power	KEE/Lisp	ART-Ada
SSM/PMAD	MSFC	Electrical Power	CLOS/Lisp	?
TEXSYS	ARC/JSC	Thermal Control	MTK/KEE/Lisp	?
PWS	MSFC	ECLSS	CLIPS/C	ART-Ada

Table 2-1: Selected Advanced Automation Projects for SSF

2.2 Requirements for Future Projects

Several prototype expert systems have been developed for testbed environments at various NASA centers. Most of these systems are developed in Lisp environments initially, and some of them are being ported to Ada environments using ART-Ada. We believe that future expert systems for the Station should be designed with the following requirements:

- A clear migration path should be defined so that they be deployed on the ground-based control center initially but be migrated to the Space Station eventually.

- Distributed and cooperative processing is essential. Consider the following scenario:

  1. Initially, all modules of an expert system are deployed on the ground.

  2. As they mature, some modules are migrated to the Station.

  In step 2, some modules are on the Station and some are on the ground, and distributed, cooperative processing occurs between the onboard modules and the ground modules. It is also important to note that expert systems in the control center would never become obsolete due to the migration. The ground controllers would have to use expert systems regardless of the migration.

- Ada should be used as early as possible during the project so that minimum effort be wasted porting from one environment to another.

The architecture proposed in this report will address these issues.

3. Intelligent Real-Time Problem Solving Architectures

In 1989, the Air Force launched a new research initiative focused on Intelligent Real-Time Problem Solving (IRTPS). As part of IRTPS phase II, a workshop was held in Santa Cruz, California, November 6 and 7, 1989. In the preface of the workshop report, the goal of this research initiative is explained [Erman 90].

> The state of the art in real-time AI systems lags far behind the needs of the applications requiring this technology. Current researchers view this technology as being at least one generation behind the expectations for real-time applications, the computational requirements of which are increasing in complexity faster than the speed of today's processors. Current problem-solving technology does not meet the expected stringent requirements for predictable and high quality results achieved in a timely manner in the presence of excessive demands for resources, where response time and computing resources are limited and varying, focus of attention shifts frequently, large amounts of information must be managed under severe time constraints, data and knowledge uncertainties exist, and goals conflict.
>
> Many current and proposed programs throughout DoD and the Air Force will require Intelligent Real-Time Problem Solving (IRTPS) capabilities. As a consequence, the Air Force needs to increase the national focus on IRTPS throughout the computer science community, including those segments involved with AI approaches to problem solving, software engineering for traditional real-time systems, decision analysis, and control theory.

We believe that the Space Station Freedom program, and other manned and unmanned programs throughout NASA also require IRTPS capabilities. Although the problem domains are different from those of the Air Force, many NASA applications also have to deal with issues related to real-time problem solving cited above.

3.1 Current IRTPS Architectures

In this section, selected architectures for IRTPS are summarized mainly from the IRTPS workshop report [Erman 90].

3.1.1 DVMT

[Erman 90] summarizes DVMT as follows:

> The Distributed Vehicle Monitoring Testbed has been developed at the University of Massachusetts since 1978. Its blackboard architecture has been extended with incremental planning control. This architecture extends tradi-

> tional blackboard systems by incorporating a sophisticated planner to control the problem-solving activity. The planner uses approximate processing techniques to hypothesize possible solutions to pursue/refute, and plans knowledge source actions to efficiently perform this pursuit/refutation. This planning can incorporate time constraints by replacing time-consuming actions with less time-consuming (or no) actions, to achieve problem-solving deadlines at the price of reduced solution quality. The control mechanism are built on top of the GBB (Generic Blackboard) shell, available from the University of Massachusetts.

See [Lesser et al 88] for details.

3.1.2 Guardian

Guardian is a patient monitoring and diagnosis system in a surgical intensive care unit [Hayes-Roth 89], [Hayes-Roth et al 89], [Ash & Hayes-Roth 90], [Washington & Hayes-Roth 90], [Washington et al 90], [Washington & Hayes-Roth 89], [Collinot & Hayes-Roth 90]. It is based on BB1 [Hayes-Roth 85], [Garvey et al 87], [Hewett 88]. Guardian runs in Common Lisp on multiple TI Explorers. [Hayes-Roth 89] describes a generic IRTPS architecture in the Guardian system as follows:

> The proposed architecture is a blackboard architecture, whose key features include: distribution of perception, action, and cognition among parallel processes, limited-capacity I/O buffers with best-first retrieval and worst-first overflow, dynamic control planning, dynamic focus of attention, and a satisficing execution cycle. Together, these features allow an intelligent agent to trade quality for speed of response under dynamic goals, resource limitations, and performance constraints.

3.1.3 Phoenix

Phoenix is described in [Cohen et al 89] and also summarized in [Erman 90] as follows:

> Phoenix is a real-time, adaptive planner that manages forest fires in a simulated environment. Alternatively, Phoenix is a search for the functional relationships among the designs of agents, their behaviors, and the environments in which they work. In fact, both characterizations are appropriate and together exemplify a research methodology that emphasizes complex, dynamic environments and complete, autonomous agents. The Phoenix group, at the University of Massachusetts, is empirically exploring the constraints the environment places on the design of intelligent agents.... The Phoenix system comprises five levels of software: Discrete Event Simulator (DES), Map, Basic Agent Architecture, Agents and Agent Organization. The DES creates the illusion of a continuous world, where natural processes and agents are acting in parallel, on serial hardware. The Map level contains the

data structures that represent the current state of the world as perceived by agents as well as the "world as it really is" and the methods that update the state of the world. The basic agent architecture provides the structure for sensors, effectors, reflexes and problem solving capabilities. The agent level describes the agents that we have designed for the Phoenix environment of forest fire fighting. The organization level includes a hierarchical organization of agents in which one fireboss directs multiple agents. The Phoenix environment (the DES and map level), the basic agent architecture and the agents with organization are independent software packages available for other researchers. The system runs in Common Lisp with Flavors on a TI Explorer.

3.1.4 PRS

PRS is described in [Georgeff & Ingrand 89a] and [Georgeff & Ingrand 89b] and also summarized in [Erman 90] as follows:

> The Procedural Reasoning System (PRS) is a generic architecture for real-time reasoning and acting that has been developed at SRI. PRS is capable of operating efficiently in continuously changing environments. It can both perform goal-directed reasoning and react rapidly to unanticipated changes in its environment. It includes meta-level reasoning capabilities, which can be tailored by the user, employing the same language used to describe domain-level reasoning. PRS has been applied to various tasks, including malfunction handling on the NASA space shuttle, threat assessment, and the control of an autonomous robot.

3.1.5 RT-1

RT-1 is described in [Dodhiawala et al 89] and also summarized in [Erman 90] as follows:

> RT-1 is a small-scale, coarse-grained, distributed, event-driven architecture based on the blackboard paradigm. It consists of a collection of reasoning modules which share a common blackboard data space and operate asynchronously. Each reasoning module has 3 processes: the I/O process for inter- and intra-module communication, the blackboard demon process for asynchronous blackboard maintenance operations, and the reasoning process which performs the knowledge processing actions for the reasoning module. The main features of the reasoning process are: multiple, prioritized event channels for improved responsiveness to more critical events, explicit meta-level reasoning capability which permits opportunistic performance between the extremes of completely reactive to completely goal-directed according to changing workload and priorities. Thus, RT-1 addresses the responsiveness, timeliness, and graceful adaptation aspects of real-time as defined in [Dodhiawala et al 89]. RT-1 has associated with it measures and metrics that help evaluate the performance of the system.

3.1.6 SOAR

SOAR is described in [Laird et al 87] and also summarized in [Erman 90] as follows:

> Soar is an AI architecture that combines a recognition-driven memory (a production system), the ability to interact directly with sensors and effectors, a decision cycle driven by the system's knowledge and perceptions, the ability to automatically generate subgoals, and the ability to learn from experience. Soar is written in Common Lisp, and runs on a variety of machines (Sun 3 & 4, DEC VAX & 3100, IBM RT, TI Explorer, ...), in Common Lisps provided by several different vendors.

We have implemented a subset of Soar capabilities in ART-Ada. The learning capability called chunking was not included in the ART-Ada implementation. Only a small example (monkeys and bananas) was tested in this implementation. A translator was built to translate the original Soar programs to the ART-Ada version. The translator does not complete the translation, however. The output of the translator must be modified manually to complete the translation.

Soar is an interesting AI problem solving architecture, but the current architecture does not address any real-time issues. Although we are aware of the research effort to develop a newer version of Soar based on neural networks (a.k.a. Neuro-Soar), we have not seen any publications on Neuro-Soar. If available, Neuro-Soar would probably address many real-time issues which have not been addressed by the current version.

3.1.7 Summary

Selected IRTPS architectures are summarized in table 3-1. As shown in this table, all IRTPS architectures are implemented in Lisp.

System	Organ.	Application	Implement.
DVMT	UMass	Vehicle Monitoring	GBB/Lisp
Guardian	Stanford	Patient Monitoring	BB1/Lisp
Phoenix	UMass	Fire Fighting	Flavors/Lisp
PRS	SRI	Space Shuttle Monitoring	Lisp
RT-1	FMC	Pilot's Associate	GBB/Lisp
SOAR	CMU	Many	Lisp

Table 3-1: Selected IRTPS Architectures

3.2 Problems in Current Architectures

As pointed out in the introduction, most existing IRTPS architectures are based on Lisp. Lisp-based implementations usually require garbage collection, which is not good for real-time systems. Research activities in conventional real-time areas focus on more conventional programming languages such as Ada rather than Lisp.

Most of these architectures are based on distributed processing. Agents are often distributed over multiple processors. The reasoning component of these architectures, however, is usually located on a single processor. For example, although the Guardian system is distributed over multiple TI Explorers, its reasoning component is a blackboard-based system that resides in a single TI Explorer.

In the next chapter, we will propose a distributed agent architecture (DAA) that overcomes these shortcomings and meets requirements for SSF advanced automation projects discussed in the previous chapter.

4. Distributed Agent Architecture

4.1 Technical Objectives

A distributed agent architecture (DAA) is designed to support real-time knowledge-based systems for the Space Station Freedom. DAA has the following technical objectives:

- The overall system performance should satisfy real-time requirements. Onboard systems should prevent catastrophic failures during the absence of assistance from ground-based systems due to the malfunction of communication systems.

- Onboard systems should adapt gracefully to dynamic environments by trading quality for speed of response.

- The architecture should be based on distributed and cooperative processing, which will enable migration of knowledge-based system modules from ground-based systems to onboard systems.

- Its baseline implementation language should be Ada. Ada will make it possible to employ traditional real-time computing methodologies and to deploy knowledge-based systems in embedded systems. If both ground systems and onboard systems are implemented in Ada, it would be easier to migrate modules from ground to the Station. Inference has, previously, implemented an Ada-based expert system tool, ART-Ada, to facilitate the deployment of expert systems in Ada [Lee & Allen 89], [Lee & Allen 90].

4.2 Design Overview

The distributed agent architecture (DAA) for real-time knowledge-based systems is depicted in figure 4-1. Other architectures that are similar to DAA are Guardian [Hayes-Roth et al 89], [Hayes-Roth 89] and Phoenix [Cohen et al 89]. Although the idea of classifying agents into two categories --- reactive and cognitive --- is inspired by Phoenix, DAA is more closely modeled after Guardian mainly because the domain of the proposed DAA testbed application (APEX) is closer to that of Guardian's. There are differences between Guardian and DAA, however:

- Although Guardian is distributed over multiple processors, its reasoning component based on BB1 [Hayes-Roth 85] runs on a single processor. In DAA, the reasoning component itself is distributed over multiple agents; reactive agents and cognitive agents. Reactive agents may be physically separated from cognitive agents by thousands of miles.

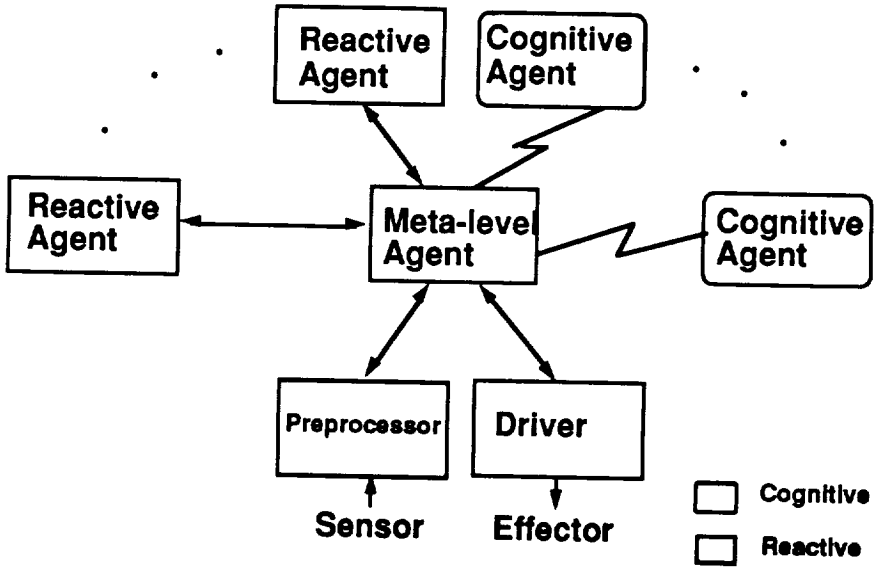


Figure 4-1: Distributed Agent Architecture for real-time knowledge-based systems

- The proposed implementation language for the DAA testbed is Ada while Guardian is implemented in Common Lisp. It is possible, however, that C- or Lisp-based implementations are included as cognitive agents in DAA.

4.3 Reactive Agents

Reactive agents are designed to meet hard real-time requirements. Hard real-time requirements are different from soft real-time in that if hard deadlines are not met, catastrophic failures are likely to occur. Catastrophic failures include the loss of human lives, the loss of major hardware components, etc. On the other hand, even if soft deadlines are violated, no major catastrophic failures are likely to occur.

It is also critical that reactive agents fit into embedded processors of the Space Station Freedom. Some AI tasks can be directly implemented in a procedural language such as Ada. The use of Ada will enable us to take advantage of recent progress that has been made in the area of real-time computing in Ada. A noteworthy example is the rate monotonic theory that can guarantee schedulability based on analytical methods [Sha 89], [Sha & Goodenough 90].

The rate monotonic theory guarantees schedulability of multiple tasks if certain conditions are satisfied. There are some restrictions, however:

- The execution time of a task must be known because it is a parameter in conditions that must be satisfied.

- It assigns the highest priority to a periodic task with the shortest period. Therefore, it prevents tasks from having priorities based on other criteria.

- The theory applies only to multiple tasks --- periodic and aperiodic --- that reside on a single processor. SEI (Software Engineering Institute) is currently working on the multi-processor version of the rate monotonic theory.

It is not clear whether the theory can be used for dynamic scheduling. It is usually used before the program execution to determine whether deadlines could be met. If deadlines are not met, periods of periodic tasks must be adjusted properly. We believe that the theory can be used to adjust periods dynamically if they are allowed to change dynamically. The theory does not prescribe how to find periods that would meet the deadlines, however.

With the right Ada runtime executive that supports rate monotonic scheduling, the schedulability can be guaranteed in advance by applying the theory analytically. It is expected that the Ada 9X Project will incorporate the rate monotonic algorithm in the next revision of the Ada language, which is due for release in 1993.

An AI technique that is useful for reactive agents is approximate or "anytime" reasoning. For example, Guardian uses a Bayesian belief network to provide reactive diagnosis. Each node of a Bayesian belief network is associated with an action. When a deadline is reached, Guardian simply recommends the action associated with the current node. If more time is given, it will continue to refine its belief and may recommend a conflicting action later on. We plan to implement an approximate reasoning module based on Bayesian belief networks in Ada.

Fuzzy logic-based systems [Lim & Takefuji 90], [Zadeh 88], [Togai & Watanabe 86] can also be used as reactive agents, using either modeling software or fuzzy hardware. In fact, fuzzy logic may subsume probabilistic reasoning using Bayesian belief networks. Fuzzy systems are becoming popular in Japan [Schwartz 90]. Togai InfraLogic, Inc in Irvine, California manufactures fuzzy-system chips and modeling software written in C. Fuzzy systems are suitable for reactive agents because:

- Real-time response can be achieved by implementing the logic on a chip.

- Fuzzy logic allows approximate reasoning.

Various reactive planning methods have been proposed [Agre & Chapman 87], [Drummond 88], [Kaelbling 88], [Nilsson 89], [Schoppers 87]. These planning methods (a.k.a. universal planning) have been sharply criticized mainly for the exponen-

tial growth of their size with the complexity of the domain [Ginsberg 89]. We plan to study both sides of arguments and investigate the possibilities of implementing reactive planning agents using some of these methods in DAA.

4.4 Cognitive Agents

Cognitive agents are traditional knowledge-based systems that are designed to meet soft real-time requirements. AI problems such as diagnosis demand accuracy of solution within a soft deadline rather than sacrifice of solution quality to meet a hard deadline. While reactive agents address the latter through approximate reasoning, cognitive agents should be based on AI techniques that facilitate deeper reasoning. For example, in Guardian, model-based reasoning is used for cognitive diagnosis while a Bayesian belief network is used for reactive diagnosis.

Although AI systems usually run on a ground-based engineering workstation today, it is becoming increasingly important that these systems are readily available in real-time embedded environments. Some examples are:

- The Pilot's Associate (PA) project for automation of the military combat aircraft cockpit,
- The Submarine Operations Automation System (SOAS) project, and
- NASA's manned and unmanned space programs.

Inference has already developed ART-Ada, an Ada-based expert system tool, for this specific purpose. ART-Ada supports rule-based reasoning as well as frame-based reasoning that can be used to implement model-based reasoning. When the current version of ART-Ada is used, the total memory requirement for an ART-Ada application with hundreds of rules is 2-3 megabyte. It may be reasonable for embedded systems based on newer processors such as the Intel 80386 and 80960, the Motorola 68000 and 88000, and the MIPS RISC chip. It is important, however, to note that the current version of ART-Ada is not optimized. The primary focus of the current release was to provide functionality. Inference plans to release an optimized version of ART-Ada in the near future.

While Ada compilers are improving, they still have not reached the maturity of C compilers. In fact, because of numerous bugs found in the Ada compilers used for this project, we could not make some of the obvious performance optimizations that could have made ART-Ada faster and smaller [Lee 90]. For example, the bit-level representation clause and pragma pack in the Verdix Ada compiler prevented us from optimizing the size of data structures.

In addition to compiler problems, we also discovered some fundamental issues with the

Ada language itself that also affected the performance of ART-Ada [Lee 90]. In particular, the problem with dynamic memory management has the most significant impact on the execution size and performance of ART-Ada.

Due to the dynamic nature of expert systems, it is necessary to allocate memory dynamically at runtime in ART-Ada and ART-IM. The direct use of *new* and *unchecked_deallocation* is the only dynamic memory management method available in Ada. The problem with this method is that *new* incurs a fixed overhead associated with each call and it is called very frequently to allocate a relatively small block for an individual data structure. It results in a performance penalty in size and the slower execution speed. This is also aggravated by the poor implementation of *new* in the Ada compiler.

The existing Ada features, *new*, *unchecked_deallocation*, and *unchecked_conversion*, are too restrictive and totally inadequate for a complex system that requires efficient memory management. More flexible features (perhaps in addition to the existing ones) should be provided. This is particularly important in embedded system environments that impose a severe restriction on the memory size. We believe that these issues should be considered for the Ada 9X standard. In fact, they have been presented to several members of the Ada 9X Project in a meeting held in Washington, D.C. in March, 1990. However, the revised Ada language will not be available until 1993 or later.

Our future research effort will be focused on implementing ART-Ada's own memory manager using an existing technology. If it is not possible to implement it in Ada, we will implement it in an assembly language. ART-Ada has an Ada code generator, which generates Ada code that relies on new and unchecked_deallocation. We will have to redesign the code generator to make it compatible with the new memory manager.

One of the Pilot's Associate teams lead by McDonnell Aircraft Company has successfully used Inference's Lisp-based expert system tool (ART™) in the earlier phases. They are currently evaluating ART-Ada in consideration for the next phase. As part of the evaluation, they have proposed some enhancements for ART-Ada to improve real-time support. These enhancements are mainly in the area of dynamic rule scheduling. A brief discussion of a wish list compiled by McDonnell Aircraft Company and a proposal for real-time extensions to ART-Ada is included in Appendix II.

4.5 A Meta-Level Agent

In a distributed architecture like DAA, the problem is how to provide meta-level control and coordination between distributed agents. A meta-level agent can be viewed as a common blackboard for meta-level control and coordination. Some examples of meta-level control are:

- to control the data input rate of the preprocessor --- when a serious problem arises, the input data rate can be reduced so that agents spend more resources in dealing with the current situation;

- to assign tasks to agents --- crisis situations may have to be handled by reactive agents to provide quick fixes while cognitive agents may follow up on it later;

- to reconcile conflicting recommendations --- when reactive agents and cognitive agents make conflicting recommendations, it is necessary to reconcile the differences; and

- to schedule operations for effectors --- when multiple agents try to control effectors, it is necessary to schedule effector assignments.

Another important area of coordination is timeline management. Following [Ash & Hayes-Roth 90], we intend to implement three timelines where each timeline records one type of information. The *occurred* timeline is used for representing facts acquired from monitoring sensors. The *expected* timeline represent what we expect in the future. The *intended* timeline represents *goals*. The intended timeline is different from the expected timeline in that actions can be taken to ensure that goals are met, whereas no actions need to be taken to produce expected results. Any agents can process or post *events* in any timelines through the meta-level agent. We intend to use ART-Ada to implement the meta-level agent.

4.6 Interagent Communication

There are several possible layers in the interagent communication protocol:

- protocol for interprocess communication,

- protocol for telemetry,

- protocol for distributed objects,

- protocol for distributed knowledge bases, and

- protocol for distributed autonomous agents.

Unix interprocess communication protocol (e.g. sockets and TCP/IP) would be a reasonable low-level protocol for prototypes. We intend to develop a protocol for distributed objects because we believe that it is an optimal layer for interagent communication. Other higher-level protocols are interesting research topics, but they may not be as practical as the distributed object protocol. Eventually, protocols used in

prototypical systems should be replaced with actual protocols supported by the Space Station Freedom.

4.7 APEX Testbed

The proposed testbed for DAA is a real-time monitoring and diagnosis expert system called APEX (Autonomous Power EXpert) for the electrical power distribution system of the Space Station Freedom [Truong et al 89], [Walters et al 90]. We will use APEX to illustrate how DAA can be applied to real-time knowledge-based systems for Space Station Freedom. It was previously implemented in KEE and Common Lisp and is being ported to ART-Ada and Ada at NASA Lewis Research Center. The APEX testbed will be used to demonstrate the advantages of this approach.

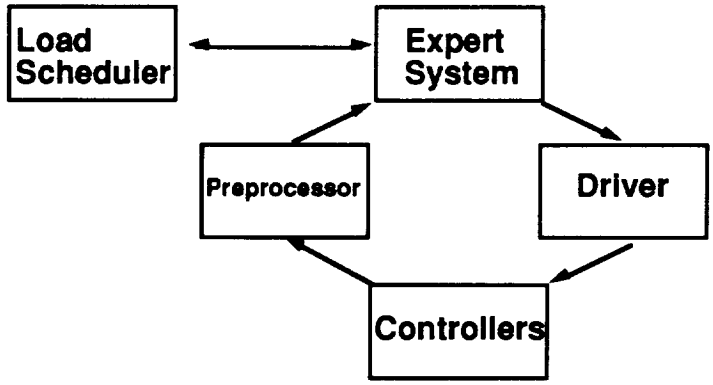


Figure 4-2: Current APEX

Figure 4-2 is a simplified block diagram of the current APEX implementation while Figure 4-3 is that of the new implementation based on DAA. In the current implementation of APEX, there are three modules:

- an expert system module written in KEE and Common Lisp that detects multiple faults, predicts possible future faults, and recommends fixes;
- a scheduler module written in C based on linear programming that schedules

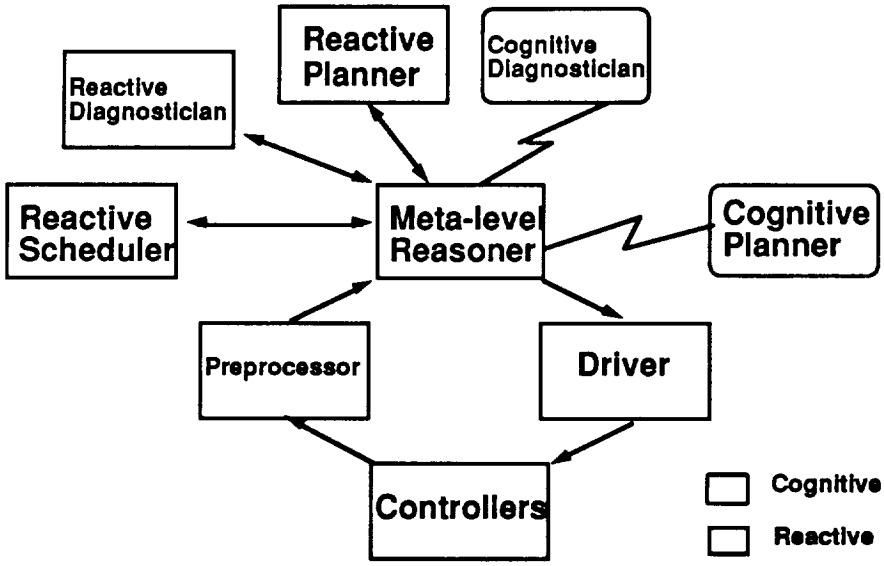


Figure 4-3: APEX based on Distributed Agent Architecture

electrical power distribution for maximum utilization of generated electrical power; and

- several software controller modules written in Ada that detect single faults and fix them immediately [Wright et al 89].

The software controller modules are written in Ada and deployed on the hardware controllers of the electrical power distribution system. These modules are designed to meet timing requirements of less than a second. They are examples of reactive agents.

The scheduler module is implemented separately from the expert system module, and runs on a PC communicating through a network. It is expected to be deployed on the Station as a reactive agent because its absence is unacceptable when the transmission between the Station and the control center is down. This module seems to lack dynamic scheduling capability. We intend to investigate the possibilities of applying AI techniques for dynamic scheduling. NASA Lewis Research Center is also considering COMPASS (COMPuter Aided Scheduling System). COMPASS is an interactive planning and scheduling system developed by McDonnell Douglas, and is available through NASA Johnson Space Center [Bayer 89]. It is written in Ada and uses X windows interfaces.

The expert system module should be distributed; more critical functionality that requires reactive responses should be separated as a reactive diagnostician and deployed on the Station while less critical functionalities such as trend analysis and long-term prediction can remain as a cognitive diagnostician in the ground-based control center. Following [Hayes-Roth et al 89], [Hayes-Roth 89], the reactive diagnostician based on *associative* reasoning methods will be implemented as a Bayesian belief network while the cognitive diagnostician based on rule- and model-based reasoning methods will be implemented in ART-Ada. By the same token, a recovery planner may have to be separated into a reactive planner and a cognitive planner. It is our intention to investigate the possibilities of adopting reactive planning methods found in various literatures [Agre & Chapman 87], [Drummond 88], [Kaelbling 88], [Nilsson 89], [Schoppers 87] to implement a reactive planner.

5. Future Work

Our future work will focus primarily on the implementation of the distributed agent architecture (DAA) for real-time knowledge-based systems for NASA's Space Station Freedom Program. The proposed testbed for DAA is a real-time monitoring and diagnosis expert system called APEX (Autonomous Power EXpert) for the electrical power distribution system of the Space Station Freedom [Truong et al 89], [Walters et al 90]. It was previously implemented in KEE and Common Lisp and is being ported to ART-Ada and Ada at NASA Lewis Research Center. This expert system, however, is not designed with the proposed architecture in mind. The main focus of the research is to identify candidate components in APEX for distributed processing and to facilitate cooperative processing between these components to meet the real-time requirements of the SSF electrical power system. Since APEX has been developed under NASA sponsorship, we assume that we will be granted permission to utilize it in this work.

Another assumption is that the underlying implementation language will be Ada and Ada-based expert system tools. Ada has been mandated for many projects of government agencies, including DoD, NASA and FAA, in which AI technology is expected to play a significant role. For this reason, it is critical to address AI problems in Ada environments. An Ada-based expert system tool under consideration is ART-Ada. ART-Ada is a proprietary software system developed and marketed by Inference. Since APEX, the proposed testbed for this project, is already being ported from KEE to ART-Ada by NASA Lewis Research Center, we intend to use ART-Ada. Cognitive agents, some reactive agents, and the meta-level agent of DAA will be implemented in ART-Ada. Other reactive agents and communications packages will be directly implemented in Ada for deployment on the 80386-based processors with four megabytes of memory that are standard processors on the Station. It is also possible to integrate C- and Lisp-based implementations as cognitive agents in DAA although it has been mandated that all new software systems for the Station should be implemented in Ada. In Appendix I, ART-Ada is further described, and other Ada-based tools are also summarized.

The critical part of our future effort will be to evaluate feasibility and benefits of DAA in the real-time environment of the APEX testbed and to identify issues for eventual deployment in the operational real-time environments of the Space Station Freedom. The expectation is that the APEX testbed experience will lead directly into a workable, usable implementation of the distributed agent architecture in the context of NASA's Space Station Freedom project. The most obvious application is to apply DAA to other prototype expert systems currently being developed for the Station such as OMS. The end system would need to be integrated with other systems on the Station and deployed in the operational environment of the Station.

6. Conclusion

The "Real-Time Expert Systems" project for NASA Johnson Space Center, under a subcontract to the University of Houston - Clear Lake, has provided several valuable insights into NASA's Space Station Freedom advanced automation projects.

- Requirements for SSF advanced automation projects have been identified.
- A new architecture based on distributed agents have been proposed to meet those requirements.
- Possible testbed applications for the distributed agent architecture have been recommended.

A successful application of the distributed agent architecture will yield new insights into distributed and cooperative processing between multiple heterogeneous agents in real-time environments. It will serve as a framework, or model, for more general applications of distributed real-time AI architectures.

References

[Agre & Chapman 87]
Agre, P., Chapman, D.
Pengi: An Implementation of a Theory of Activity.
In *Proceedings of the International Conference on Artificial Intelligence.* AAAI, 1987.

[Ash & Hayes-Roth 90]
Ash, D., Hayes-Roth, B.
Temporal Representations in Blackboard Architectures.
Technical Report KSL 90-16, Knowledge Systems Laboratory, Stanford University, March, 1990.

[Barrios 89a] Barrios Technology, Inc.
CLIPS/Ada Advanced Programming Guide.
Barrios Technology, Inc., 1989.

[Barrios 89b] Barrios Technology, Inc.
CLIPS/Ada Architecture Manual.
Barrios Technology, Inc., 1989.

[Bayer 89] Bayer, S.E.
Space Station Freedom Program Capabilities for the Development and Application of Advanced Automation.
Technical Report MTR-89W00279, The MITRE Corporation, December, 1989.

[Cohen et al 89] Cohen, P.R. et. al.
Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments.
AI Magazine 10(3), Fall, 1989.

[Collinot & Hayes-Roth 90]
Collinot, A, Hayes-Roth, B.
Real-Time Control of Reasoning: Experiments with Two Control Models.
Technical Report KSL 90-17, Knowledge Systems Laboratory, Stanford University, March, 1990.

[Dewberry 89] Dewberry, B.S.
The Environmental Control and Life Support System Advanced Automation Project, Phase 1 - Application Evaluation.
In *Proceedings of the Workshop on Space Operations Automation and Robotics.* NASA Johnson Space Center, July, 1989.

[Dodhiawala et al 89]
Dodhiawala, R. et. al.
Real-Time AI Systems: A Definition and An Architecture.
In *Proceedings of the International Joint Conference on Artificial Intelligence.* IJCAI, 1989.

[Dominick et al 89]
Dominick, J. et. al.
NASA Systems Autonomy Demonstration Project: Advanced Automation Demonstration of Space Station Freedom Thermal Control System.
In *Proceedings of the Workshop on Space Operations Automation and Robotics.* NASA Johnson Space Center, July, 1989.

[Drummond 88] Drummond, M.
Situated Control Rules.
In *Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems.* University of Rochester, 1988.

[Erman 90] Erman, L.D. (editor).
Intelligent Real-Time Problem Solving (IRTPS): Workshop Report.
Technical Report TTR-ISE-90-101, Cimflex Teknowledge Corp., January, 1990.

[Filman et al 89] Filman, R.E., Bock, C., and Feldman, R.
Compiling Knowledge-Based Systems Specified in KEE to ADA.
Technical Report Final Report, NASA Contract NAS8-38036, IntelliCorp Inc., August, 1989.

[Garvey et al 87] Garvey, A., et. al.
BB1 User Manual - Common LISP Version 2.0.
Technical Report KSL 86-61, Knowledge Systems Laboratory, Stanford University, August, 1987.

[Georgeff & Ingrand 89a]
Georgeff, M.P., Ingrand, F.F.
Decision-Making in an Embedded Reasoning System.
In *Proceedings of the International Joint Conference on Artificial Intelligence.* AAAI, 1989.

[Georgeff & Ingrand 89b]
Georgeff, M.P., Ingrand, F.F.
Monitoring and Control of Spacecraft Systems Using Procedural Reasoning.
In *Proceedings of the Space Operations Automation and Robotics Workshop.* NASA Johnson Space Center, 1989.

[Giarratano 89] Giarratano, J.C.
CLIPS User's Guide.
NASA Johnson Space Center, 1989.

[Ginsberg 89] Ginsberg, M.L.
Universal Planning: An (Almost) Universally Bad Idea.
AI Magazine 10(4), 1989.

[Hardin & Albin 88]
Hardin, D.S., Albin, K.L.
Taking Inference to Task.
In *Proceedings of the Conference on Artificial Intelligence & Ada.*
Department of Computer Science, George Mason University, November, 1988.

[Hayes-Roth 85] Hayes-Roth, B.
A Blackboard Architecture for Control.
Artificial Intelligence 26(3), July, 1985.

[Hayes-Roth 89] Hayes-Roth, B.
Architectural Foundations for Real-Time Performance in Intelligent Agents.
Technical Report KSL 89-63, Knowledge Systems Laboratory, Stanford University, December, 1989.

[Hayes-Roth et al 89]
Hayes-Roth, B. et. al.
Intelligent Monitoring and Control.
In *Proceedings of the International Joint Conference on Artificial Intelligence.* IJCAI, 1989.

[Hewett 88] Hewett, M.
BB1 User Manual - Version 2.1 Update (Common LISP).
Technical Report KSL 86-61a, Knowledge Systems Laboratory, Stanford University, February, 1988.

[Hirshfield & Slack 88]
Hirshfield, S.H., Slack, T.B.
ERS: An Expert System Shell Designed and Implemented in Ada.
In *Proceedings of the Conference on Artificial Intelligence & Ada.*
Department of Computer Science, George Mason University, November, 1988.

[Inference 87] Inference Corporation.
Ada-ART, Specification for an Ada-based State-of-the-Art Expert System Construction Capability.
Technical Report, Inference Corporation, August, 1987.

[Inference 89a] Inference Corporation.
ART/Ada Design Project - Phase I, Final Report.
Technical Report, Inference Corporation, March, 1989.

[Inference 89b] Inference Corporation.
ART-Ada/VMS 2.0 Beta Reference Manual.
Inference Corporation, 1989.

[Ishida 88] Ishida, T.
Optimizing Rules in Production System Programs.
In *Proceedings of the National Conference on Artificial Intelligence.* AAAI, 1988.

[Jaworski et al 87]
Jaworski, A., LaVallee, D., Zoch, D.
A Lisp-Ada Connection for Expert System Development.
In *Proceedings of the Third Annual Conference on Artificial Intelligence & Ada.* Department of Computer Science, George Mason University, October, 1987.

[Jouchoux et al 87]
Jouchoux, A. et. al.
Developing a Spacecraft Monitor and Control System in Ada.
In *Proceedings of the Joint Ada Conference, National Conference on Ada Technology and Washington Ada Symposium.* 1987.

[Kaelbling 88] Kaelbling, L.P.
Goals as Parallel Program Specification.
In *Proceedings of the National Conference on Artificial Intelligence.* AAAI, 1988.

[Kelly et al 89] Kelly, C. et. al.
The Migration of an Expert System Application from Lisp to Ada.
In *Proceedings of the Conference on Artificial Intelligence & Ada.* Department of Computer Science, George Mason University, November, 1989.

[Labhart & Williams 89]
Labhart, J., Williams, K.
Ada MeriTool: A Software Tool for Knowledge-Based Systems.
In *Proceedings of the Conference on Artificial Intelligence & Ada.* Department of Computer Science, George Mason University, November, 1989.

[Laird et al 87] Laird, J.E., Newell, A. and Rosenbloom, P.S.
SOAR: An Architecture for General Intelligence.
Artificial Intelligence 33(1), 1987.

[Lee 90] Lee, S.D.
Toward the Efficient Implementation of Expert Systems in Ada.
In *Submitted to the TRI-Ada Conference.* ACM, 1990.

[Lee & Allen 89] Lee, S.D., Allen, B.P.
Deploying Expert Systems in Ada.
In *Proceedings of the TRI-Ada Conference.* ACM, 1989.

[Lee & Allen 90] Lee, S.D., Allen, B.P.
ART-Ada Design Project - Phase II, Final Report.
Technical Report, Inference Corporation, February, 1990.

[Lesser et al 88] Lesser, V.R., Pavlin, J., Durfee, E.H.
Approximate Processing in Real-Time Problem Solving.
AI Magazine 9(1), Spring, 1988.

[Lim & Takefuji 90]
Lim, M.H., Takefuji, Y.
Implementing Fuzzy Rule-Based Systems on Silicon Chips.
IEEE Expert 5(1), February, 1990.

[Lukefahr et al 89]
Lukefahr, B.D. et. al.
A Diagnostic Prototype of the Potable Water Subsystem of the Space Station Freedom ECLSS.
Technical Report UAH Research Report Number 824, Johnson Research Center, University of Alabama, Huntsville, November, 1989.

[Martin 89] Martin, J.L.
A Development Tool for Real-Time Expert Systems.
Alsynews 3(1), March, 1989.

[NASA/JSC 89] Artificial Intelligence Section, NASA Johnson Space Center.
CLIPS Version 4.3 Reference Manual.
NASA Johnson Space Center, 1989.

[Nilsson 89] Nilsson, N.J.
Action Networks.
In *Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems.* University of Rochester, 1989.

[Schoppers 87] Schoppers, M.J.
Universal Plans for Reactive Robots in Unpredictable Domains.
In *Proceedings of the International Joint Conference on Artificial Intelligence.* IJCAI, 1987.

[Schwartz 90] Schwartz, T.J.
Fuzzy Systems Come to Life in Japan.
IEEE Expert 5(1), February, 1990.

[Sha 89] Sha L., Goodenough J.B.
Real-Time Scheduling Theory and Ada.
Technical Report CMU/SEI-89-TR-14, Carnegie-Mellon University, Software Engineering Institute, April, 1989.

[Sha & Goodenough 90]
Sha, L., Goodenough, J.B.
Real-Time Scheduling Theory and Ada.
Computer 23(4), April, 1990.

[Simonian 88] Simonian, R.P., Crone, M.
InnovAda: True Object-Oriented Programming in Ada.
The Journal of Object-Oriented Programming 1(4), November/December, 1988.

[SPS 88] Software Productivity Solutions, Inc.
Classic-Ada User Manual.
Software Productivity Solutions, Inc, 1988.

[Stockman 88] Stockman, S.P.
ABLE: An Ada-Based Blackboard System.
In *Proceedings of the Conference on Artificial Intelligence & Ada.* Department of Computer Science, George Mason University, November, 1988.

[Taylor & Hansen 89]
Taylor, E.W., Hanson, M.A.
OMS FDIR - Initial Prototyping.
In *Proceedings of the Workshop on Space Operations Automation and Robotics.* NASA Johnson Space Center, July, 1989.

[Togai & Watanabe 86]
Togai, M., Watanabe, H.
Expert System on a Chip: An Engine for Real-Time Approximate Reasoning.
IEEE Expert 1(3), Fall, 1986.

[Truong et al 89] Truong, L., et. al.
Autonomous Power Expert Fault Diagnostic System for Space Station Freedom Electrical Power System Testbed.
In *Proceedings of the Workshop on Space Operations Automation and Robotics.* NASA Johnson Space Center, July, 1989.

[Wallnau et al 88]
Wallnau, K.C. et. al.
Construction of Knowledge-Based Components and Applications in Ada.
In *Proceedings of the Conference on Artificial Intelligence & Ada.* Department of Computer Science, George Mason University, November, 1988.

[Walls & Lollar 89]
Walls, B., Lollar, L.F.
Automation in the Space Station Module Power Management and Distribution Breadboard.
In *Proceedings of the Workshop on Space Operations Automation and Robotics.* NASA Johnson Space Center, July, 1989.

[Walters et al 90]
Walters, J.L., et. al.
Autonomous Power Expert System.
In *Proceedings of the Goddard Conference on Space Applications of Artificial Intelligence.* NASA Goddard Space Flight Center, May, 1990.

[Washington & Hayes-Roth 89]
Washington, R., Hayes-Roth, B.
Input Data Management in Real-Time AI Systems.
In *Proceedings of the International Joint Conference on Artificial Intelligence.* IJCAI, August, 1989.

[Washington & Hayes-Roth 90]
Washington, R., Hayes-Roth, B.
Abstraction Planning in Real-Time.
Technical Report KSL 90-15, Knowledge Systems Laboratory, Stanford University, March, 1990.

[Washington et al 90]
Washington, R., et. al.
Using Knowledge for Real-Time Input Data Management.
Technical Report KSL 90-14, Knowledge Systems Laboratory, Stanford University, March, 1990.

[Wright 89]
Wright, P.A.
Ada Real-Time Inference Engine (ARTIE).
In *Proceedings of the Conference on Artificial Intelligence & Ada.* Department of Computer Science, George Mason University, November, 1989.

[Wright et al 89] Wright, T, Mackin, M., Gantose, D.
Development of Ada Language Control Software for the NASA Power Management and Distribution Testbed.
Technical Report, NASA Lewis Research Center, 1989.

[Zadeh 88] Zadeh, L.A.
Fuzzy Logic.
Computer 21(4), April, 1988.

I. Ada-Based Expert System Tools

In recent years, an increasing number of Ada-based AI tools have been reported in publications. And yet only a handful of commercial off-the-shelf (COTS) tools are available today. In this appendix, we will review both commercial and non-commercial Ada-based tools.

I.1 Commercial Tools

In this section, we will review Ada-based tools that are generally available as commercial off-the-shelf (COTS) tools.

I.1.1 ART-Ada

Inference has been involved with Ada-based expert systems research since 1986. Initial work centered around a specification for an Ada-based expert system tool. The result of this research activity is summarized in [Inference 87]. In 1988, the ART-Ada Design Project was initiated to design and implement an Ada-based expert system tool. At the end of Phase I of this project, a working prototype was successfully demonstrated. This research activity is reported in [Inference 89a] and [Lee & Allen 89]. In 1989, during the ART-Ada Design Project - Phase II, the Phase I prototype was extended and refined so that it could be released to beta sites [Lee & Allen 90]. At the end of 1989, ART-Ada was released to beta sites as ART-Ada 2.0 Beta on the VAX/VMS and Sun/Unix platforms [Inference 89b]. In 1990, eight beta sites, four NASA sites and four Air Force sites, will be evaluating ART-Ada 2.0 for eight months by developing expert systems and deploying them in Ada environments.

The objectives of the ART-Ada Design Project were two fold:

1. to determine the feasibility of providing a hybrid expert system tool such as ART in Ada, and

2. to develop a strategy for Ada integration and deployment of such a tool.

Both of these objectives were met successfully when ART-Ada 2.0 beta was released to the beta sites. During the evaluation period, the following objectives are important:

1. to evaluate any bugs or performance problems, and

2. to determine any issues related to particular embedded system environments.

ART-Ada allows applications of a C-based expert system tool called ART-IM to be deployed in various Ada environments. While ART-IM's inference engine was reimple-

mented in Ada, ART-IM's front-end (its parser/analyzer and graphical user interface) was reused as the ART-Ada development environment. The ART-IM kernel was enhanced to generate Ada source code that would be used to initialize Ada data structures equivalent to ART-IM's internal C data structures, and also to interface with user-written Ada code. Once the development is complete, the application is automatically converted to Ada source code. It is then compiled and linked with the Ada runtime kernel, which is an Ada-based inference engine. The overview of ART-Ada is depicted in figure I-1.

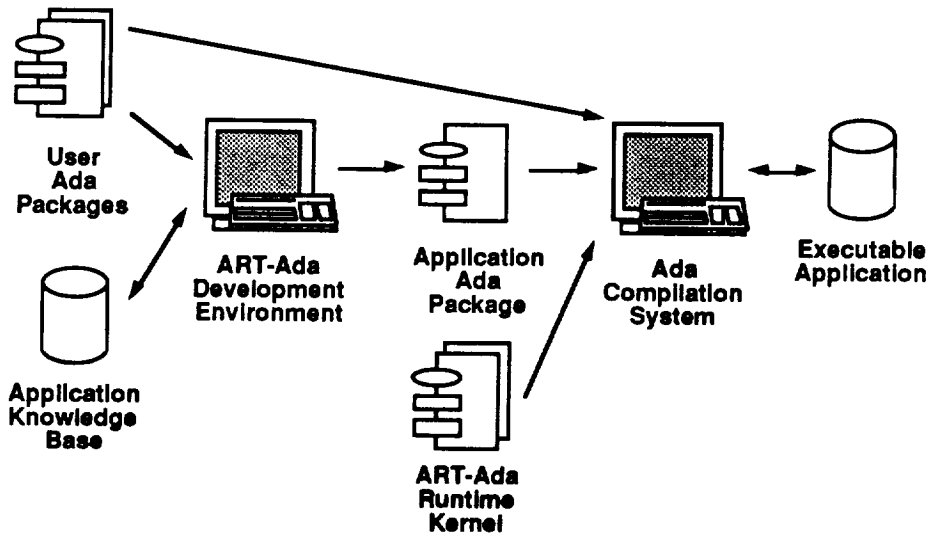


Figure I-1: Overview of ART-Ada

ART-Ada provides comprehensive functionality for knowledge representation and inferencing within an open architecture that facilitates embedding knowledge base components into broader software applications.

For knowledge representation, ART-Ada combines object-oriented and relational data modeling with independent rules to express decision logic. Objects are represented as symbols with associated attributes and attribute values. Relationships tie together objects, literal values, and structured values. Objects can be organized into classification hierarchies through inheritance.

A rule in ART-Ada is a condition paired with an associated action. The condition identifies those combinations of objects and relationships to which the action should be

applied. Actions can augment or modify the data model (e.g., to derive a logical inference), and can perform procedures such as input/output. Rules are frequently thought of as expressing "IF condition THEN action". However, a more accurate way to think of rules is "WHENEVER data satisfies condition APPLY actions to data". The critical point is that rules are independent; i.e., each rule is a free standing statement that fully expresses the conditions under which it is applicable. This is in contrast with an "IF-THEN" statement of a procedural programming language, which is explicity executed as a consequence of the flow of control within a program. The conditions under which an IF-THEN statement is reached in the flow of control comprise significant implicit conditions on when and how to perform the THEN-part of the statement.

I.1.2 CHRONOS

CHRONOS is a commercial expert system tool written in Ada [Martin 89]. It is developed by a French company, Euristic Systems and marketed mainly in France.

I.1.3 Classic-Ada

Classic-Ada is an object-oriented programming language developed and marketed by Software Productivity Solutions, Inc [SPS 88]. Its input language is based on Smalltalk, but it works like C++; it is a preprocessor that generates Ada source code. Unlike ART-Ada, the generated Ada code is self-sufficient; it does not require an Ada runtime kernel to compile it. Although Classic-Ada does not support rules, its object-oriented programming capability is similar to that of ART-Ada.

I.1.4 CLIPS/Ada

CLIPS (C Language Interface Production System) is a C-based forward-chaining rule-based expert system tool developed by NASA Johnson Space Center [NASA/JSC 89], [Giarratano 89]. Its syntax is very close to those of ART, ART-IM and ART-Ada. CLIPS has been ported to Ada by Barrios Technology for NASA Johnson Space Center [Barrios 89a], [Barrios 89b]. Unlike ART-Ada, which uses ART-IM's development environment to generate Ada source code, CLIPS/Ada is a complete reimplementation of CLIPS. CLIPS/Ada does not support object-oriented programming, a truth maintenance system, and explanation generation utilities, all of which are supported in ART-Ada. Forward-chaining rules and facts are the only knowledge representation methods available in CLIPS/Ada.

I.1.5 Summary

Commercial Ada-based tools are summarized in table I-1.

System	Organ.	Graphical UI	Rules	OOP	Platforms
ART-Ada	Inference	DECwindows/C	Yes	Yes	VAX/VMS,Sun
CHRONOS	Euristic	Windows	Yes	No	PC/AT,VAX/VMS,Unix
Classic-Ada	SPS	?	No	Yes	VAX/VMS,Sun,etc.
CLIPS/Ada	Barrios	None	Yes	No	VAX/VMS

Table I-1: Commercial Ada Tools

I.2 Research Tools

In this section, we will review Ada-based tools that are not generally available as a commercial off-the-shelf (COTS) tool. Some tools are still being developed, and others are being used internally. Most of these tools are reported in the proceedings of the annual conference on artificial intelligence & Ada (AIDA), 1987-1989.

I.2.1 ABLE

ABLE (Ada BLackboard Environment) is being developed by Boeing Military Airplanes and is based on Erasmus, Boeing's Lisp-based blackboard system [Stockman 88]. It attempts to translate Erasmus source code into Ada so that blackboard applications developed in Erasmus could be deployed in Ada. Since Lisp code is embedded within the knowledge sources of Erasmus, a Lisp-to-Ada translator must be developed. We believe, however, that it is difficult, if not impossible, to develop such a translator that can handle the full Lisp language. Currently, the translator exists only in the most primitive state. Another issue is integration with existing Ada packages while an Erasmus application is developed in a Lisp environment. Integration of Lisp with Ada is not generally feasible.

I.2.2 ARTIE

Boeing Military Airplanes developed ARTIE (Ada Real-Time Inference Engine), an Ada implementation of Pascal Inference Engine (PIE) [Wright 89]. It appears to be designed specifically for the 1750A architecture. To run on the 1750A, software modules must fit in the 64K segment. From the beginning, the requirement was that its run-time overhead be no greater than 64K. It achieves this goal by executing with only 35 Kbytes of run-time overhead. Its inference engine is a simple forward-chaining rule system with no support for object-oriented programming. It has been ported to Apollo, VAX, Sun and Silicon Graphics.

I.2.3 ERS

ERS (Embedded Rule-based System) is a C-based expert system tool used internally by PAR Government Systems. Recently, ERS has been ported to Ada [Hirshfield & Slack 88]. Its design is based on that of the AL/X system, a rule-based consultation system, which incorporates a probabilistic inference mechanism. Its inference engine uses a version of Nilsson's goal-directed backward-chaining algorithm for determining the degree of belief in the goal hypotheses.

I.2.4 FLAC

FLAC (Ford Lisp-Ada Connection) uses a Lisp environment on a Lisp machine to develop an expert system application and generate Ada code [Jaworski et al 87]. In FLAC, the knowledge base is specified using a graphical representation similar to that of VLSI design (e.g. OR gates and AND gates). Since FLAC's development environment is based on Lisp, it probably does not support Ada call-in/call-out. FLAC's knowledge base is pre-compiled and static, which means that objects may not be added or deleted dynamically at runtime although their values may be changed. This imposes major functionality restrictions that do not exist in ART-Ada.

I.2.5 KEE/Ada

IntelliCorp has done some research to develop a system for translating KEE applications into Ada [Filman et al 89]. On the surface, the main difficulties of the approach seem to be Ada integration during development and the translation of Lisp code to Ada. The advantage of ART-Ada is that Ada subprograms can be called directly from the knowledge base during development. Since the development environment of ART-Ada is written in C, Ada call-back is used to integrate Ada subprograms. Ada call-back simply means that the Ada *main* program calls a C program (ART-Ada development environment), which calls back to Ada subprograms. This is the only proper way to call Ada from another language such as C. Ada is not only a programming language but also a runtime environment. The use of the Ada *main* program ensures the proper initialization of the Ada runtime environment.

The problem with KEE is that it is written in Lisp. Lisp is also a runtime environment like Ada. Therefore, it would be hard to start Lisp from the Ada main, which is the only way to call back to Ada from Lisp. In fact, the Lucid Common Lisp 3.0 used to implement KEE supports call-out to C, Fortran, and Pascal, but not Ada.

ART-Ada also supports Ada call-in. We developed an Ada binding --- Ada packages that interface with the ART-Ada development environment, which is implemented in C. ART-Ada provides over 200 ART-Ada functions that can be called from Ada to control and access the knowledge base procedurally. It would be impossible to implement an Ada binding for KEE in Lisp so that KEE's Lisp functions could be called from Ada.

In KEE, since neither Ada call-out nor call-in is available, actions in the rule right-hand side (RHS) must be implemented in Lisp. The automatic translation of the Lisp code to Ada would alleviate the burden of manual translation if it is technically feasible. It might be possible to translate a small subset of Lisp to Ada automatically. Even so, the efficiency of the translated Ada code would not be as good as hand-written Ada code. This approach also excludes the use of existing Ada packages (for numerical analysis, signal processing, etc.) in the knowledge base. In ART-Ada, existing Ada packages can be easily integrated directly into the knowledge base even during development.

I.2.6 MeriTool

MeriTool was originally implemented in C and ported to Ada by Merit Technology Incorporated [Labhart & Williams 89]. Its inference engine uses the Merit Enhanced Traversal Engine (METE), which is similar to Forgy's RETE algorithm. Its rule language syntax is also similar to that of OPS5; it supports objects with attribute-value pairs. It does not appear, however, to support object-oriented programming such as inheritance, methods, etc.

I.2.7 Other Related Work

InnovAda is an object-oriented programming language developed by Harris Space Systems Corporation and is similar to Classic-Ada [Simonian 88].

Unisys' RLF (Reusability Library Framework) project focuses on the development of Ada software components supporting two knowledge representation techniques --- structured inheritance networks and rule-based inference systems --- and the domain specific reuse of these components [Wallnau et al 88].

Developed by Collins Government Avionics Division of Rockwell International, SLATE is an inference engine written in Ada and targeted for embedded applications, specifically avionics [Hardin & Albin 88]. SLATE employs Ada tasking to decouple inference and search.

II. A Proposal for Real-Time Extensions to ART-Ada

II.1 Introduction

In recent years, expert systems have been widely used to monitor and control various systems in real-time. These systems require capabilities to process real-time data in a timely manner and to reason about temporal data and relations. When timing requirements cannot be met using a single processor, it may be necessary to distribute the system among multiple processors. In this case, an efficient communication protocol must be provided. In order to respond to a new situation effectively, decision strategy may have to be changed dynamically.

To meet these requirements, we propose a framework for building real-time expert systems as an extension to an Ada-based expert system tool, ART-Ada. The framework is based on the following features:

- a set of tools for performance monitoring and tuning,
- temporal reasoning and trend analysis,
- communication protocol for passing messages between distributed expert systems, and
- dynamic rule scheduling.

We believe that this framework, when used in conjunction with ART-Ada, will simplify and systematize the development of real-time expert systems.

II.2 Performance Monitoring and Tuning

The performance of an expert system varies widely depending on how it is implemented. It is often necessary to monitor activities in the pattern matcher (e.g. the number of pattern instantiations, partial matches, activations, etc.) or the execution time of a rule RHS (right-hand side) action in order to determine areas for optimization. Performance analysis can be aided by a set of tools that graphically display the information.

Unlike conventional software, rule-based systems are sensitive to the ordering of patterns in rules. Currently, the only way to optimize pattern ordering is to monitor activities in the pattern and join networks and optimize them manually. It may be possible, however, to automate this manual optimization process. It has been reported that an automated tool was successfully used to optimize join ordering [Ishida 88]. An op-

timization algorithm can be automatically applied to a rule-based program to find near-optimal pattern ordering for the entire program.

II.3 Temporal Reasoning and Trend Analysis

In a real-time expert system, it is often necessary to reason about and perform statistical analysis on *temporal* data -- data that change over time. In order to avoid information overloading, several levels of abstraction should be used. Raw data should be preprocessed to suppress noises and redundant data. Historical data should not participate in the pattern-matching process directly. Rather, high-level abstraction acquired by applying temporal reasoning and trend analysis to the historical data, should used in the knowledge base.

For example, a monitoring and control expert system for the Space Shuttle called INCO uses a layered approach [Bayer 89]:

> A layered architecture was developed to move from raw telemetry data through generic data conversion algorithms to procedural techniques for isolating algorithmic domain specific knowledge and finally to rule-based techniques for heuristic knowledge. An interesting characteristic of this layered approach is that, as data moves up the layers, the total amount of data decreases, but the information value of the data increases. For example, 192,000 bits of information enter at the first layer, but the rule-based expert system operates only 350 facts generated by the layer three algorithms.

We propose to implement a set of functions that can be layered on top of ART-Ada as a separate library for temporal reasoning and trend analysis. This library is based on the concepts, *monitors*, *events* and *timers*, which are explained below.

II.3.1 Monitors

A *monitor* is used to store historical data in a ring buffer outside of the knowledge base. A monitor is refered to only by its name, which is stored in a hash table. The following is a partial list of functions related to a *monitor*:

- (make-monitor *name* &optional *action*) returns *monitor*
- (set-monitor-polling *second*) returns T or NIL
- (set-monitor-size *number*) returns T or NIL
- (get-monitor-value *monitor time*) returns *value*
- (get-monitor-min *monitor start-time end-time*) returns *value*

- (get-monitor-max *monitor start-time end-time*) returns *value*
- (get-monitor-average *monitor start-time end-time*) returns *value*
- (increasing-p *monitor start-time end-time*) returns T or NIL
- (decreasing-p *monitor start-time end-time*) returns T or NIL
- (steady-p *monitor start-time end-time*) returns T or NIL
- (changed-p *monitor*) returns T or NIL

The following example creates monitors for voltage, current, and temperature, and analyzes the trend of these parameters.

```
;;; Telemetry data comes in every second and
;;; up to 100 values per item should be saved.
(set-monitor-polling 1)
(set-monitor-size 100)

;;; create monitors and set alarm actions
(make-monitor voltage voltage-alarm)
(make-monitor current current-alarm)
(make-monitor temperature temperature-alarm)

;;; the trend of the voltage is increasing
;;; over the last 5 seconds
(increasing-p voltage -5 0)

;;; the minimum value of the battery's voltage
;;; over the last 30 seconds > 30
(> (set-monitor-min voltage -30 0) 30)

;;; always (Voltage = 0) over the last 5 seconds
(= (get-monitor-min voltage -5 0)
   (get-monitor-max voltage -5 0))
```

II.3.2 Events

Events are used to extract temporal relations between parameters. *Events* are a collection of time that satisfies certain conditions specified in a function, *make-events*. The following is a partial list of functions related to *events*:

- (make-events *monitor predicate value start-time end-time*) returns *events*
- (and-events *events1 events2*) returns *events*
- (or-events *events1 events2*) returns *events*

- (before-p *events1 events2*) returns T or NIL
- (after-p *events1 events2*) returns T or NIL
- (while-p *events1 events2*) returns T or NIL

The following example examines temporal relations between parameters defined in the previous example.

```
;;; ever (Voltage > 5) during over the last 10 seconds
(make-events voltage > 5 -10 0)

;;; (Voltage > 0) before (Temperature < 10)
;;; in last 10 seconds
(before-p (make-events voltage > 0 -10 0)
          (make-events temperature < 10 -10 0))

;;; (Voltage = 2) while (Temperature > 5)
;;; in last 10 seconds
(while-p (make-events voltage = 2 -10 0)
         (make-events temperature > 5 -10 0))

;;; (Voltage > 0 and Current = 1) before
;;; (Temperature = 0) in last 10 seconds
(before-p (and-events (make-events voltage > 0 -10 0)
                      (make-events current = 0 -10 0))
          (make-events temperature = 0 -10 0))
```

II.3.3 Timers

Rule-based systems are usually data-driven. In a real-time system, however, processing must be driven by time as well as data. A *timer* can be used to implemented time-driven processing. The following is a partial list of functions related to a *timer*:

- (set-timer *name type hour min sec* &optional *action*) returns T or NIL where *type* is **at** or **every** or **offset**
- (reset-timer *name*) returns T or NIL
- (advance-time *hour min sec*) returns T or NIL
- (set-time *hour min sec*) returns T or NIL

In the following example, the message, lunch-message, will be invoked automatically at noon.

```
(def-art-fun lunch-message ()
  (printout t "lunch time everyone..." t))

(set-timer lunch-time at 12 0 0 lunch-message)
```

II.4 Dynamic Rule Priority

In real-time AI architectures, the priority of a task should be dynamically determined based on the timing constraints and the resource requirements of the task [Hayes-Roth et al 89], [Dodhiawala et al 89]. In the current version of ART-Ada, the priority of a rule cannot be changed dynamically. If the priority of a rule is allowed to be changed at runtime, rule scheduling strategy can also be modified dynamically.

In the following example, the closer the distance is, the higher priority will be assigned to the rule activation. In fact, the same rule can be activated with different priorities if its priority can be modified dynamically. In order for the rule dynamic priority to function properly, the priorities of all activated rules in the *agenda* must be refreshed before a rule is selected for execution.

```
(defrule foo
  (declare (salience ?s = 1/?d))
  (schema ?enemy-plane (distance ?d))
  =>
  (...))
```

If the execution time of a rule is known, it can be used to adjust its priority. It is often desirable to assign a higher priority to a rule with a shorter execution time. In fact, it is the strategy used by the rate monotonic theory [Sha 89], [Sha & Goodenough 90]. In the following example, duration is the execution time of a rule RHS action. The execution time can be either measured or estimated.

```
(defrule foo
  (declare (duration 1 sec))
  (...)
  =>
  (...))
```

II.5 Message Passing between Distributed Expert Systems

Multiple cooperating ART-Ada applications can run on loosely-coupled multiple processors. ART-Ada supports object-oriented programming. A *method* is a function associated with an object or a class that can be inherited. When a message is sent via an ART-Ada function *send*, an appropriate method will be invoked. If objects are distributed over multiple processors, and a data dictionary is used to define mapping between a processor and an object, the message passing mechanism through *send* can be used without modification to implement distributed message passing. When a message is sent, the system can simply check the data dictionary and send the message to the appropriate processor. Each ART-Ada application can use an asynchronous function to check its message queue between every rule firing.

II.6 Other Issues

In this section, a wish list compiled by the Pilot's Associate project team at McDonnell Aircraft Company, St. Louis, Missouri is included without modification for completeness:

1. access to rule scheduler
2. context dependent salience
3. automatic data time tagging
4. concurrent working memory updates
5. working memory data locking
6. optional separation of rule schedule/execute
7. pattern/join network access
8. ability to add demons to assert/retract, etc.
9. user defined rule declarations (duration, etc.)

Issues #2 and #9 are addressed by the proposed extensions to ART-Ada discussed above.

The current version of ART-Ada already addresses the issue #6. ART-Ada stores assertions and retractions specified in a rule RHS action in a queue for further processing during the execution of a rule RHS. After the RHS action is fully executed, the assertion/retraction queue is processed.

Issues #1, #3, #7, and #8 are technically feasible. It would be a good idea to include these features in ART-Ada.

Issues #4 and #5, on the other hand, may be difficult to implement. Working memory update is usually done through assertions and retractions, which gets stored in a queue and processed sequentially. The code segment for assertions and retractions is a critical section; it must be executed sequentially to maintain consistency. The issue #5 (working memory data locking) is related to concurrent update. If concurrent update is not feasible, working memory data locking may not be necessary.

III. Feedback from Review Meetings

III.1 Introduction

A series of meetings were held in Houston, Texas to review the distributed agent architecture (DAA) proposed in this paper during May 2-4, 1990. Most attendees are currently working on the projects related to the Space Station Freedom Program. These meetings are summarized below.

III.2 Meeting with the Communications and Tracking Group

Most of the attendees are working on the onboard software for the SSF communications and tracking system. They do not expect expert system tools such as CLIPS/Ada to qualify for flight mainly because they cannot be validated and verified. Another problem is onboard resource limitation. They do not expect onboard processors and memories on the Space Station to be upgraded from the current configuration in the near future. It is also unlikely that special processors like fuzzy chips would be allowed on the Station.

The current configuration of the standard processor is based on the Intel 80386 with four megabytes of memory. It is expected that the operating system would consume up to two megabytes of memory. Therefore, only two megabytes of memory are available for application software. Realistically, however, only one megabyte or less would be available to a particular application. For example, memory requirement for OMA is currently only 500 Kbytes, which is not sufficient for CLIPS-based applications. Their approach was to extract algorithms out of their CLIPS-based prototype and to reimplement it directly in Ada using a tool called FIBS (Fault Isolation by Bit Strings). Other software (e.g. OMA) may have to follow the same path.

They are concerned about the design of OMA and its interfaces to other Tier II Managers. According to them, the current OMA design permits software to disable manual overrides of some system operations. They believe, however, that the manual override should always be allowed. They are also concerned about coordination with the ground-based system for communications and tracking.

It was pointed out that graceful degradation should have been included in the DAA design. Graceful degradation can be achieved by using only a small portion of available resources (e.g. memory) during normal operation. This design principle permits minimal performance degradation when resources become tighter. Graceful degradation is a principle often used in real-time systems.

Another interesting point was that on the Space Station, the same processor used for

diagnosis would also have to acquire necessary data for diagnosis. It would not have a luxury to use a separate processor for preprocessing, which is usually the case for ground-based systems.

DAA proposes a distributed object protocol as an optimal layer for interagent communication. As pointed out during this meeting, the DMS protocol already includes a distributed object protocol, RODB, which confirms the DAA design.

Attendees were:

- Oron Schmidt, NASA/JSC, EE7
- David Overland, NASA/JSC, EE7
- Mark Glorioso, NASA/JSC
- Joseph Kahan, MDAC and NASA/JSC, MOD, DE32
- John Bandlow, MDAC and NASA/JSC, MOD, DE32

III.3 Meeting with MITRE

MITRE plans to port their CLIPS-based OMA prototype to Ada using ART-Ada. They would like their new prototype based on ART-Ada to be used as a starting point for the production version of OMA.

They are interested in DAA. They are not aware of any other architectures like DAA proposed for the Space Station. They would like to see it further developed.

As pointed out during the meeting, the Ada mandate for the Space Station applies only to new software components. Existing software components written in other languages can be reused. Lisp may be an exception and may not be allowed.

Attendees (all from MITRE) were:

- Jim Spitzer, Department Head, Space Data Systems
- Chris Marsh
- Debra Schreckenghost
- Kathryn Cooksey
- Dona Erb

- Bruce Treea
- Stu Bezl
- Audrey Dorofee
- Debbie McGrath

III.4 Meeting with Ford Aerospace and IBM

The plan for the SSCC (Space Station Control Center) is to baseline the present MCCU, which is only a couple of years old. The Mission Support Contract, which is used to build the SSCC, will be reviewing the functional specification for the SSCC in November, 1990. The functional spec is being developed now by UDeF (User Defined Function) requirements. The people in the UDeF group are mostly shuttle flight controllers. Some of them will be transferred to the SSCC eventually.

According to the IBM and Unisys engineers, there would be no advanced automation in the initial SSCC. However, they must make sure that their design would not prevent any future advanced automation in the SSCC.

According to them, OMGA no longer exists. Instead, its functionality has been distributed over multiple subsystems and integrated into the SSCC. The communication interfaces are still centralized.

A possible area for advanced automation in the SSCC is the existing MCC databases, FAPS, SCAPS and MOMS, that are used to troubleshoot problems. They are presently not very useful due to the cumbersome retrieval and browsing mechanisms.

Attendees were:

- Matt Hansen, Ford Aerospace
- James M. Lee, Unisys
- Rodney W. Holden, IBM
- Charles Copeland, Ford Aerospace
- Amadeo Montemayor, Ford Aerospace
- John Engvall, AI Center Manager, Ford Aerospace
- Troy Heindel, NASA, MOD

III.5 Meeting with McDonnell Douglas

The current memory requirement for OMA is only 500 Kbytes. The estimated size of their OMA design based on CLIPS/Ada is 1.2 Mbytes, which may be too optimistic and could be larger in reality. ART-Ada's memory requirement is probably similar to that of CLIPS/Ada.

The primary objective of the current release of ART-Ada was to provide the functionality. Optimization was not the main concern. The OMA developers are very interested in the optimized version of ART-Ada, preferrably on a 386-based processor. They just had an OMA PDR, and they have another year before CDR, during which they would like to evaluate ART-Ada. Inference is looking for funding to optimize ART-Ada and to create a new version before the end of 1990. In the meantime, Inference would like to arrange an evaluation copy so that they could evaluate the current version of ART-Ada.

Attendees were:

- Jack Dean, MDAC, OMA
- Dennis Lawler, NASA, ED, ISB
- Don Woods, MDAC, Advanced Automation
- Steven Domingue, MDSS-SSD
- Steven Jowers, MDSSC-ESD
- Colin Clark, MDSSC-ESD
- Robert Faltisco, MDSSC-SSD
- Deborah Conley, MDSSC-SSD

III.6 Conclusion

In general, DAA was well received by all attendees. In particular, DAA focuses on the cooperation between onboard systems and ground-based ones, which is not currently well addressed by the Space Station Freedom Program. It is not easy to achieve cooperative processing between onboard systems and ground systems. We believe that it is technically feasible, but it is difficult because it involves multiple organizations. Currently, onboard systems (e.g. OMA) are handled by Work Package contractors while ground-based systems (e.g. OMGA) are handled by MSC contractors. If an architecture like DAA is adopted as a general framework for the Space Station, it could be used as a "glue" between different contractors.

Many flight-related software components will reside in the SSCC because onboard computing resources are very limited. We believe that ground-based flight-related software systems should operate in the same environment as onboard flight software for two reasons:

- If ground-based software components are crucial for flight, it should be considered as part of the flight software. The same verification and validation standard that is normally applied to onboard flight software should also be applied to these software components.

- If ground-based software components are destined to migrate to the Station, it would be essential for the SSCC to have the same operating environment as the onboard environment.

Because of these reasons, the Ada mandate should be imposed on the development of any new ground-based flight-related software components as well as onboard software.

Another important issue raised by DAA is the assessment of risks caused by communication delays. Average communication delay may be less than a minute in normal operating conditions, which is not significant. On the other hand, there might be longer delays caused by "blind spots" in the communication networks or by hardware failures in the transmission systems. NASA should assess any risks of having catastrophic failures on the Station due to the absence of support from ground-based systems during these communication delays.