N91-20688

A Framework for Building Real-Time Expert Systems

S. Daniel Lee

Inference Corporation
550 N. Continental Blvd.
El Segundo, CA 90245

Abstract

NASA's Space Station Freedom is an example of complex systems that require both traditional and AI real-time methodologies. It has been mandated that Ada should be used for all new software development projects. The Station also requires distributed processing. 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.) To address these issues, we propose a distributed agent architecture (DAA) that can support a variety of paradigms based on both traditional real-time computing and artificial intelligence. 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.

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 [13], [14], [11], [15].

We propose a distributed agent architecture (DAA) that can support a variety of paradigms based on both traditional real-time computing and artificial intelligence.

2. Distributed Agent Architecture

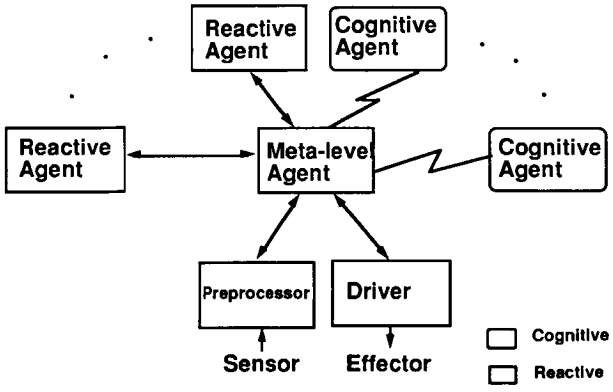

Figure 2-1: Distributed Agent Architecture

The distributed agent architecture (DAA) for real-time knowledge-based systems is depicted in figure 2-1. 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.

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 [20], [21]. AI techniques under consideration for reactive agents are approximate or "anytime" reasoning that can be implemented using Bayesian belief networks as in Guardian [8], [7]. Fuzzy logic [16], [26], [22] and reactive planning [1], [5], [10], [17], [18] 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 [2], 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.

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 [20], [21].

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.

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 [16], [26], [22] 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 [19]. 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 [1], [5], [10], [17], [18]. These planning methods (a.k.a. universal planning) have been sharply criticized mainly for the exponential growth of their size with the complexity of the domain [6]. 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. 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.

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.

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 [11]. In addition to compiler problems, we also discovered some fundamental issues with the Ada language itself that also affected the performance of ART-Ada [11]. In particular, the problem with dynamic memory management has the most significant impact on the execution size and performance of ART-Ada.

Our current research effort is 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. Another area of research is to improve real-time support in ART-Ada. Several extensions to ART-Ada are proposed to address real-time issues and included in Appendix I.

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 is 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 [2], 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.

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.

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 [23], [24]. 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.

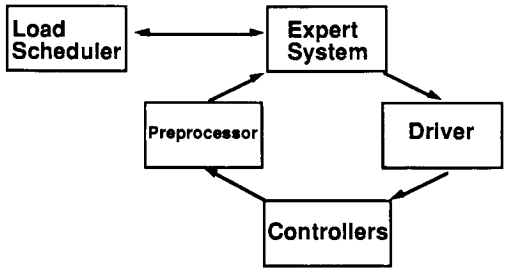

Figure 7-1: Current APEX

Figure 7-1 is a simplified block diagram of the current APEX implementation while Figure 7-2 is that of the new implementation based on DAA. In the current implementation of APEX, there are three modules:

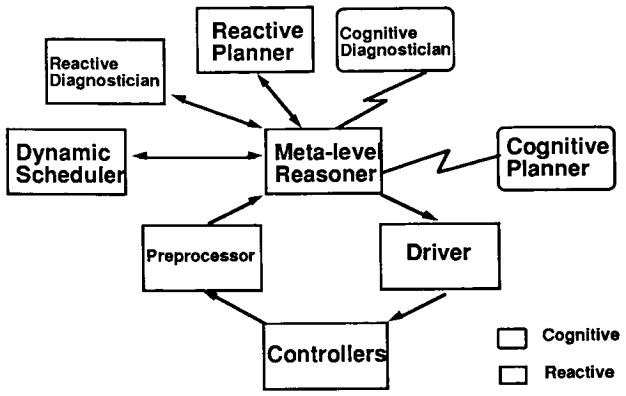

Figure 7-2: APEX based on DAA

- 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 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 [25].

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 [3]. 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 [8], [7], 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 [1], [5], [10], [17], [18] to implement a reactive planner.

8. Conclusion

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 and ground-based systems are handled by different 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 (Space Station Control Center) 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.

9. Acknowledgments

The author wishes to acknowledge the guidance and support of Chris Culbert and Bob Savely of NASA Johnson Space Center, Greg Swietek of NASA Headquarters, and Captain Mark Gersh of the U.S. Air Force. Brad Allen, Mark Auburn and Sherry Walden of Inference Corporation contributed to the project. Barbara Hayes-Roth of Stanford University, Rajendra Dodhiawala and Cindy Pickering of FMC, Francois Felix Ingrand of SRI International, Tom Broten of TRW, Rich Knackstedt and Steve Bate of McDonnell Douglas, Jerry Walters of NASA LeRC and many other NASA scientists and contractors provided useful discussions and feedback.

References

1. Agre, P., Chapman, D. Pengi: An Implementation of a Theory of Activity. Proceedings of the International Conference on Artificial Intelligence, AAAI, 1987.

2. Ash, D., Hayes-Roth, B. Temporal Representations in Blackboard Architectures. Tech. Rept. KSL 90-16, Knowledge Systems Laboratory, Stanford University, March, 1990.

3. Bayer, S.E. Space Station Freedom Program Capabilities for the Development and Application of Advanced Automation. Tech. Rept. MTR-89W00279, The MITRE Corporation, December, 1989.

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

5. Drummond, M. Situated Control Rules. Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems, University of Rochester, 1988.

6. Ginsberg, M.L. "Universal Planning: An (Almost) Universally Bad Idea". *AI Magazine 10*, 4 (1989).

7. Hayes-Roth, B. Architectural Foundations for Real-Time Performance in Intelligent Agents. Tech. Rept. KSL 89-63, Knowledge Systems Laboratory, Stanford University, December, 1989.

8. Hayes-Roth, B. et. al. Intelligent Monitoring and Control. Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 1989.

9. Ishida, T. Optimizing Rules in Production System Programs. Proceedings of the National Conference on Artificial Intelligence, AAAI, 1988.

10. Kaelbling, L.P. Goals as Parallel Program Specification. Proceedings of the National Conference on Artificial Intelligence, AAAI, 1988.

11. Lee, S.D. Toward the Efficient Implementation of Expert Systems in Ada. Submitted to the TRI-Ada Conference, ACM, 1990.

12. Lee, S.D. A Distributed Agent Architecture for Real-Time Knowledge-Based Systems. Inference Corporation, May, 1990.

13. Lee, S.D., Allen, B.P. Deploying Expert Systems in Ada. Proceedings of the TRI-Ada Conference, ACM, 1989.

14. Lee, S.D., Allen, B.P. ART-Ada Design Project - Phase II, Final Report. Inference Corporation, February, 1990.

15. Lee, S.D., Allen, B.P. ART-Ada: An Ada-Based Expert System Tool. Proceedings of the Space Operations, Applications and Research Symposium (SOAR), NASA, 1990.

16. Lim, M.H., Takefuji, Y. "Implementing Fuzzy Rule-Based Systems on Silicon Chips". *IEEE Expert 5*, 1 (February 1990).

17. Nilsson, N.J. Action Networks. Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems, University of Rochester, 1989.

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

19. Schwartz, T.J. "Fuzzy Systems Come to Life in Japan". *IEEE Expert 5*, 1 (February 1990).

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

21. Sha, L., Goodenough, J.B. "Real-Time Scheduling Theory and Ada". *Computer 23*, 4 (April 1990).

22. Togai, M., Watanabe, H. "Expert System on a Chip: An Engine for Real-Time Approximate Reasoning". *IEEE Expert 1*, 3 (Fall 1986).

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

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

25. Wright, T, Mackin, M., Gantose, D. Development of Ada Language Control Software for the NASA Power Management and Distribution Testbed. NASA Lewis Research Center, 1989.

26. Zadeh, L.A. "Fuzzy Logic". *Computer 21*, 4 (April 1988).

I. Proposed Real-Time Extensions to ART-Ada

I.1. 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 [9]. An optimization algorithm can be automatically applied to a rule-based program to find near-optimal pattern ordering for the entire program.

I.2. 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.

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*. A *monitor* is used to store historical data in a ring buffer outside of the knowledge base. A monitor is referred to only by its name, which is stored in a hash table. *Events* are used to extract temporal relations between parameters. *Events* are a collection of time that satisfies certain conditions. 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. For more details on *monitors*, *events* and *timers*, see [12].

I.3. 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 [8], [4]. 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.

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 [20], [21]. 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 (salience ?s = 1/?d))
  (declare (duration 1 sec))
  (schema ?enemy-plane (distance ?d))
  =>
  (...))
```

I.4. 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.