N93-25977

THE WORKPLACE DISTRIBUTED PROCESSING ENVIRONMENT

Troy Ames
NASA/Goddard Space Flight Center
Data Systems Technology Division
Greenbelt, Maryland 20771
(301) 286-5673
ames@kong.gsfc.nasa.gov

Scott Henderson
CTA Incorporated, Space Systems Division
6116 Executive Boulevard, Suite 800
Rockville, Maryland 20852
(301) 816-1218
scott@cta.com

Abstract

Real time control problems require robust, high performance solutions. Distributed computing can offer high performance through parallelism and robustness through redundancy. Unfortunately, implementing distributed systems with these characteristics places a significant burden on the applications programmers. Goddard Code 522 has developed *WorkPlace* to alleviate this burden. WorkPlace is a small, portable, embeddable network interface which automates message routing, failure detection, and re-configuration in response to failures in distributed systems. This paper describes the design and use of WorkPlace, and its application in the construction of a distributed blackboard system.

1. The Dilemma

WorkPlace was developed as part of the Intelligent Ground System (IGS) project within Goddard Space Flight Center's Data Systems Technology Division with funding from NASA Code R. The IGS project is exploring the use of multiple knowledge-based systems in the satellite control center, particularly in the area of platform monitoring and fault diagnosis. The current practice is to introduce isolated expert systems into operations. Our objective is to achieve a more comprehensive system that involves many expert systems that communicate and cooperate with each other and with conventional components of the control center.

We faced two technological hurdles in achieving this objective. First, we needed to provide a flexible, open mechanism for data exchange which would support multiple platforms and heterogeneous applications. Second, we needed to develop an architecture for expert systems which would accommodate the asynchronous nature of a distributed cooperative environment. This paper describes the solutions to these problems that we have developed.

2. The Blackboard Solution

Blackboard systems represent the standard metaphor for distributed problem solving in AI. That metaphor describes a team of experts who cooperate to solve some problem. These experts communicate by writing partial solutions on a blackboard. The posting of a partial solution by one expert triggers the activity of an expert with a related expertise. Together these experts progressively evolve the partial solutions into a solution to the top level problem. Thus the blackboard architecture is based on:

- a universally accessible space for posting partial solutions (the blackboard),
- partitioning of that space into multiple levels of abstraction,
- and the opportunistic application of Knowledge Sources (experts) to that space to further the current level of understanding.

Traditional implementations of the blackboard approach use shared memory within a uni-processor for the information space. When the posting of a new partial solution triggers multiple knowledge sources, conflict resolution strategies serialize the execution of those sources and their access to the blackboard. This limits performance.

Several systems have been described in the literature which provide different approaches to parallelizing blackboard systems. One family of approaches is based on the use of a multi-processor architecture. Hearsay II (Fennell and Lesser, 1976) provides a central blackboard which is written to by concurrent experts executing on a simulated multi-processor. The experts (or knowledge sources) are further de-coupled into precondition and action parts which can execute concurrently within their own copy of relevant portions of the blackboard (known as *contexts*). The central blackboard provides node

C-3

and region locks to mediate reading and writing by the concurrent knowledge sources. The CAGE system (Nii, Aiello, and Rice, 1989) also provides a central blackboard, and can be executed with inter-knowledge source, intra-knowledge source, and intra-rule parallelism on a simulated multi-processor. Similarly, CAGE provides locks to deal with concurrent reads and writes to the central blackboard. Extensive simulation experiments have been performed on this architecture to measure the relative effects of these different forms of parallelism on the performance of a representative problem. Polygon (Nii, Aiello, and Rice, 1989) departs from the central blackboard theme by parallelizing the nodes which would normally reside on the central blackboard. Direct communication between nodes obviates the need for a global data structure. Data coherence is handled through the use of "smart" slots in the nodes which decide when a new value is better than the existing value of the slot using local heuristics. Polygon also runs on a simulated multi-processor.

A second family of approaches is built on concurrent processes in one or more conventional computers communicating through Inter-process Communication (IPC) mechanisms. The transaction processing blackboard described in (Ensor and Gabbe, 1988) provides a central blackboard which mediates the interaction between satellite blackboards which operate concurrently. The central blackboard uses a transaction processing metaphor to mediate reading and writing by satellite blackboards. This system was implemented on a network of Symbolics Lisp Machines and provides a nice model of loosely coupled groups of closely coupled experts. The COPS system (Leao and Talakdar, 1988) extends the OPS5 production system to provide fact exchange between independent OPS5 processes. Remote writing is available through addressed IPC messages, but appears to be used primarily for instantiating new processes. Normal fact exchange is accomplished through the use of "ambassador" rules. Ambassador rules can be thought of as parasites which are inserted into remote COPS processes to watch for fact patterns and report detections back to the originating system. A subset of the COPS processes are designated as blackboards. These central repositories exist primarily as intermediaries between non-rule-based applications (which can not accept ambassador rules) and the other OPS5-based applications.

3. WorkPlace Architecture

Our work falls into the second family of approaches, attempting to bring the cooperation available in blackboard systems to an environment of physically distributed conventional computers. Unlike COPS and the Ensor and Gabbe system, WorkPlace places no constraints on the processing formalism used in communicating nodes, and supports a range of interfaces to TCP/IP[1]. Cooperation is built on a flexible event distribution mechanism rather than shared memory. This mechanism uses a Publish/Subscribe/Sample metaphor, providing an exceptionally simple application interface. Cast in terms of the blackboard metaphor WorkPlace offers:

- a common catalog of facts with a selectively replicated fact space,
- and parallel application of knowledge sources and transformers to that space to further the current level of understanding.

From the application's point of view there are four operations necessary to participate in the WorkPlace environment. First, the application must provide a handler for facts received over the network. The implementation of this handler is entirely up to the application. Second, the application must regularly call a ProcessEvents() function to allow the communications software to keep in contact with the rest of the group. Information destined for the application will be caught during this call and passed to the application's fact handler. Third, the application must inform the agent of its remote information needs. These needs can change dynamically throughout the life of the program. Finally, the application must explicitly make information available which might be of interest to other members of the group.

The remainder of this section explains these operations in more detail, and presents some of

1 WorkPlace currently supports UNIX and Macintosh interfaces to TCP/IP. Support for VMS may be added in the future. Intermediate blackboards are not required to integrate heterogeneous applications since no assumptions are made about the nature of those applications.

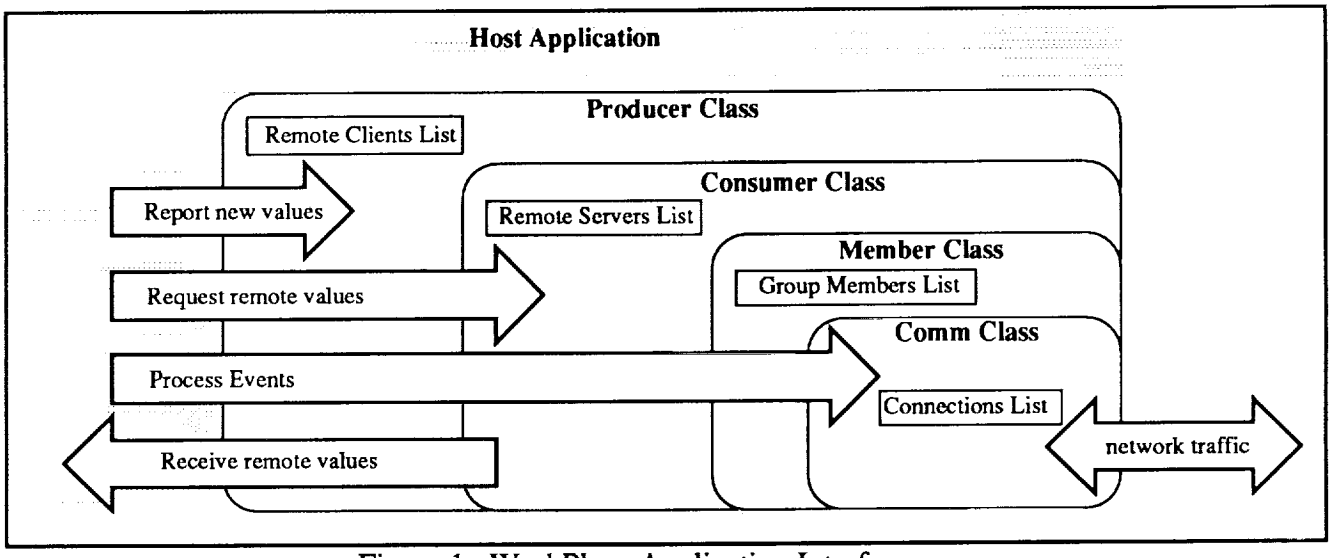

Figure 1. WorkPlace Application Interface

the ramifications of our implementation. An overview is shown in figure 1.

3.1 Membership In The WorkPlace

For applications to exchange information they must know of each other's existence and location. The list of existing applications and their locations can be thought of as membership information. WorkPlace acquires this membership information dynamically. The only static information required is the name of the group and the address of at least one member.

Dynamic membership means that the full roster and address lists are determined during the execution of the applications. The simplest approach is to use a centralized data server as an information clearing house. This server accepts connections from remote applications for either receiving or delivering information. All information produced by an application is forwarded to this data server for selective distribution to client sites. The down side of this approach is that a given piece of information is transmitted twice if a client exists for that information, and once if that information is not currently needed. The benefit is that a complete history of the products of the system is available. This centralized data server also represents a single point of failure for the environment: if the server goes down information flow stops.

The approach taken in the WorkPlace environment is to provide every application with the ability to accept and request connections from remote applications for either receiving or delivering information. The environment then becomes an association of peer nodes. That association is born with the appearance of a founding member, and ceases to exist when the last member exits. During the life of an association, any of its members, including the founding member, may leave the group and may later return. This fully distributed and dynamic design provides four benefits over centralized and static ones:

- Reduced vulnerability to individual node failures.
- Direct transmission of desired information from producer to consumer.
- No forwarding of unused information.
- The ability to add, delete, or move processes on an ad-hock basis without unnecessarily disrupting the execution of retained processes.

3.2 Product Exchange in The WorkPlace

From the application's perspective, product exchange is simply a matter of packaging information and reporting it, or receiving an information product and unpackaging it. The actual routing of information between agents occurs asynchronously with no involvement by the application. Allowing the agent to derive this information dynamically provides the flexibility necessary to accommodate changes in computational resources (e.g. processor or link failures), changes in computational load, and run

time changes to the applications suite (e.g. application crashes, upgrades to existing applications, and addition and removal of diagnostic processes).

Information is packaged as a value with a unique identifier. Identifiers are broken down further into an object specifier and an attribute specifier, as in "the float value (object specifier) temperature (attribute specifier) is 71.3 °F (value specifier)." Thus the conceptual package for an information product is a triple of the form <ObjectName>, <AttributeName>, <AttributeValue>. Services are provided for constructing attribute values from integer numbers, floating point numbers, character strings, or nested lists of these atomic types. The units in which a value is cast are assumed to be known to the receiver a-priori.

Reporting information is referred to as *Publishing* in WorkPlace. Each time the application publishes information, the agent caches the value reported. If there are registered clients for that information product, a message is forwarded to each of those clients specifying the new value. When the application publishes an information product whose identifier is different from any previously published by that application, the agent makes an announcement to all active members of the group. If the identifier has never been published by any other member of the group, the announcement identifies the object name, attribute name, and a more computationally efficient identifier for the product to be used in subsequent transactions. Otherwise the announcement simply notes the new source for that information identifier. If an application ceases to produce some information product, it can announce this fact through the *UnPublish* method. This removes the application's name from the producers list of the indicated product for every active member of the group.

The application requests remote information products by subscribing to information products. The embedded agent contacts known producers of that product and registers subscriptions with them. The agent also records the request so sources of that information which appear in the future can also be contacted. Two variants of the subscription method exist: *SubscribeToAll*, and *SubscribeToAny*. *SubscribeToAll* places subscriptions for the requested products with every agent known (now or in the future) to be capable of producing those products. When an application expects a single source for a piece of information, it can use the *SubscribeToAny* variant. SubscribeToAny places a single subscription, per product, with a random agent known to be capable of producing that product. If the selected product source stops publication of that product, quits the group, or displays anomalous behavior, then the agent will automatically move the subscription to an alternative source. A measure of fault tolerance is afforded through this mechanism by intentionally providing redundant copies of an information source on separate hardware. The flow of product updates can be halted by invoking the *UnSubscribeTo* method. This is useful when throughput disparities force the receiver to sample the data stream. An application can subscribe to and un-subscribe to a product or products an arbitrary number of times. The only overhead of *SubscribeTo* and *UnSubscribeTo* invocations is a short message to the selected supplier(s) of the product.

The application does not receive the value of a subscribed product until the value of that product changes. To obtain the current value of an information product, the application invokes the *SampleAll* or *SampleAny* agent methods. One, or more sources may or may not exist for the requested products. If no sources are known, then the supplied default value is returned to the application's product handler. If only one source is known, then a sample request is forwarded to that source. That source's agent responds to the request with the last cached value for the product. The local agent receives that value and returns it to the application by way of the application's product handler. If multiple sources are known for the requested product, then a sample request is forwarded to a randomly selected source for the "Any" case, or to each source in the "All" case.

3.3 The Network Interface

In reality the WorkPlace agent implements only the bookkeeping and protocol necessary to track group membership and information product sources and subscriptions. The actual network interface is implemented in the OSCAR (Open System for Coordinating Automated Resources) Agent class over which the WorkPlace agent is layered.

The OSCAR Agent class provides a common Application Programmer Interface (API) for network communications over several operating systems. Within this class a suite of standard routines for writing messages, reading messages, and getting connection status is defined. Subclasses implement the actual routines for specific protocols and operating systems. Currently the VMS, UNIX, and Macintosh operating systems are supported by OSCAR. New protocol implementations are added as peers in the suite of supported protocols. The OSCAR agent class selects among these protocols when a send request is made based on information it has on the location and type of the destination agent. The OSCAR agent also monitors each communication path for connection requests from remote agents.

4. Integration of a Distributed System

The IGS project has developed a testbed to evaluate and demonstrate the functionality of distributed knowledge-based systems within a control center setting (see figure 2). This testbed incorporates a spacecraft simulator, command scheduler, user interface, and three knowledge-based diagnostic systems. These testbed applications are integrated through the WorkPlace software. Our operational goal is to evolve the diagnostic components of the testbed into a platform diagnostic system for the first EOS spacecraft, due to be launched in late 1998.

The object-oriented spacecraft simulator accepts commands and generates telemetry data. A command scheduler acts as the bottleneck through which spacecraft commands are forwarded. Three knowledge-based systems interpret the telemetry stream in real time to monitor the state of spacecraft subsystems. When anomalies are detected, these systems provide explanation and advice to the user interface and optionally post suggested fixes with the command scheduler. The user interface depicts a graphical hierarchy of the spacecraft and ground components, where the user can zoom down into lower levels of detail when a problem is detected. An intelligent front end to the user interface filters and synthesizes related fault warnings to reduce information overload.

Integrating the Spacecraft Simulator

The job of the testbed is to control and monitor an object-oriented simulation of the EOS A spacecraft The model is composed of an electrical power system, thermal bus system, HIRIS (High-Resolution Imaging Spectrometer) instrument payload, and platform manager. The EOS spacecraft model is augmented by a model of the sun and the space-to-ground

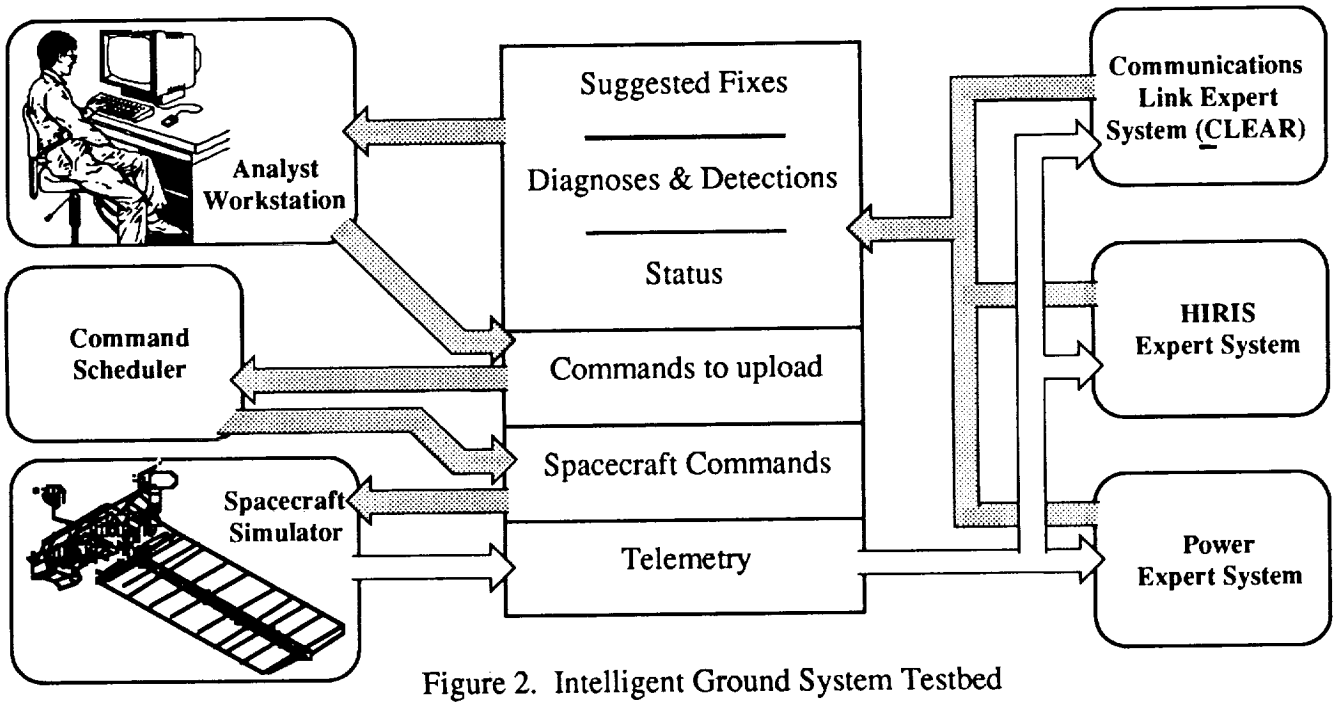

Figure 2. Intelligent Ground System Testbed

communications link. The simulation demonstrates electrical, thermal, and mechanical aspects of the spacecraft's behavior over time, with realistic responses to up-loaded commands and partial equipment failure. The simulator design is based on the connection manager architecture proposed by the Software Engineering Institute for flight simulators, as reported in (Lee, 1989), and the suggestions for object-oriented simulation in (Zeigler, 1990).

To operate successfully within the context of the testbed, the simulator needed to be able to receive commands and report telemetry. This was achieved by creating two new object types within the simulation environment. The first type received binary and serial information from other objects within the simulator, and converted and published that information as information products. These objects were then directly wired to the telemetry sources in the model. The second object type received information products and converted them to serial and binary signals. These signals were then connected to the spacecraft's command handler to allow remote control of the spacecraft. Lastly, the cyclic executive for the simulation was altered so that the embedded agent's *ProcessEvents* method could be called between simulation cycles.

Integration with Clips

The knowledge-based components of the testbed include three rule-based systems: the Communications Link Expert Assistance Resource (CLEAR), a power bus monitoring and diagnostic system (PowerFDIR), and a monitoring and diagnostic system for the spacecraft's HIRIS instrument (HirisFDIR). CLEAR was taken from an operational NASA communications fault diagnostic system. The other two systems where developed by the task expressly for the testbed.

Each of these systems is written in the "C" Language Integrated Production System (CLIPS), an expert system shell developed by Johnson Space Center (Giarrantano, 1991). We produced a distributed version of CLIPS version 5.1 (IGSClips) by embedding a WorkPlace agent. The integration required the addition of six new functions mirroring the distributed data agent methods: Publish, SubscribeToAny, SubscribeToAll, SampleAny, and SampleAll. One additional right-hand side function, GenNetSymbol, was added so that applications could generate network-unique identifiers. A call to the OSCAR ProcessEvents() function was inserted after each rule-execution cycle to service the network connection. Product updates arriving during this call are asserted into the fact base in the form (InPort <Object Name> <Attribute Name> <Attribute Value>). We have not yet evaluated the performance costs to CLIPS of the ProcessEvents call between cycles when no network information is pending.

IGSClips is similar in concept and operation to the COPS system (Leao and Talakdar 1988) described earlier. The main difference lies in the migration of group and communications management code out of the production system and into a separate module (the WorkPlace agent). Because IGSClips does not rely on ambassador rules, direct cooperation between the simulator and the diagnostic agents was possible.

The Impact of Asynchronous Operation

Two basic diagnostic architectures are present in the testbed: a shallow reactive architecture, and a deeper model-based architecture. While the deeper architectures are able to take more information into account in making their diagnoses, their integration into the asynchronous environment was more difficult.

CLEAR was implemented with the help of a domain expert who, through personal experience, was able to impart rules which related surface features (telemetry values) almost directly to diagnoses. CLEAR was ported to the testbed with only minor modifications necessary to publish its diagnoses and to format incoming telemetry to be accepted by the existing rule base.

The PowerFDIR and HirisFDIR diagnostic systems could not benefit from the compiled knowledge of a domain expert. Instead, these diagnostic systems keep two models of the spacecraft subsystem they monitor. The first model maintains the expected state of the subsystem based on a known initial state, the command stream that has been sent to the subsystem, and the known behavior of the subsystem in response to commands. The second model maintains the current state of the subsystem based on telemetry from that subsystem. Anomaly detection results from a comparison of these two models.

Temporal Coherence

Early experiments with the testbed showed that when the simulator was running on a separate platform from that which the diagnostic systems were running on, it was possible to swamp the model-based diagnostic systems with telemetry. In some cases the diagnostic systems were making recommendations for conditions which no longer existed, and in others the systems exhausted their ability to buffer incoming telemetry and crashed. Our solution was to change the diagnostic systems so that they in effect sampled the telemetry stream and then reacted to that sample. This de-coupled the processing rate of the diagnostic systems from the production rate of the telemetry source.

Unfortunately, we could not use the WorkPlace *Sample* operation to do this. The telemetry data is logically partitioned into sets which represent a snapshot of the spacecraft at a particular point in time. If the diagnostic system sampled two telemetry points which were not from the same frame, it would not be able to build a coherent picture of the current state of the reporting subsystem. Instead, when a diagnostic system wants to sample the telemetry stream, it places subscriptions for the telemetry points it needs. The diagnostic system then throws away all the telemetry it receives until the start of a new frame is detected (e.g. $element_0$ arrives). The system then caches all the subsequent data points until the end of the frame is detected, at which point the subscriptions are revoked. Now if the rule-based system is fast enough, it operates as it had previously. If at any time it is not fast enough, each diagnostic cycle only processes the most recent set of telemetry available.

This elaborate behavior on the part of the application simulates the sampling of a frame of information. If the WorkPlace agent embedded in the simulation knew that a given subset of data was part of a larger product, then that agent could take steps to guarantee the temporal coherence of the data made available to the group for sampling. At this point we have not extended the WorkPlace agent to support this, so the burden remains on the client application.

Non-monotonicity

Cooperation allows independent systems to leverage each other's expertise. A power failure on one power bus affects all the subsystems which are drawing power from that bus. The PowerFDIR has the expertise to identify the bus power failure, but the HirisFDIR does not. Through cooperation, the HirisFDIR can use external information generated by the PowerFDIR to distinguish an external power failure from an internal power distribution problem. Unfortunately, there is no guarantee that the helpful information will arrive before the subsystem monitor makes its diagnosis. The only solution we have at this time is for the subsystem monitor to retract its diagnosis when better information becomes available.

Summary

We have described our solutions to two technological hurdles standing in the way of cooperative knowledge based systems. The first, WorkPlace, provides an open system for fact exchange within a heterogeneous environment. We think that the generality of this tool makes it suitable for a wide range of applications, and that its support for "hot spares" makes it unique. Second, we have described some of the complications which have arisen from asynchrony, and how those complications have constrained the basic architecture of agents within the distributed cooperative environment. Taken together, these solutions demonstrate a viable design for physically distributed cooperative systems, and provide key tools for use in their implementation.

References

Adler, R.M. (1992). Coordinating Complex Problem-Solving Among Distributed Intelligent Agents. *1992 Goddard Conference on Space Applications of Artificial Intelligence*, CP 3141, 47-57.

Buckley, B., and Wheatcraft, L. (1992). Distributed Expert Systems for Ground and Space Applications. *1992 Goddard Conference on Space Applications of Artificial Intelligence*, CP 3141, 59-70.

Dominy, R. (1991). *Open System For Coordinating Automated Resources(OSCAR) Programmers Guide.* Technical report to NASA Goddard Space Flight Center.

Ensor, J.R., and Gabbe, J.D. (1988). Transactional Blackboards. In *Readings in Distributed Artificial Intelligence*, eds. Bond and Gasser. Morgan Kaufmann Publishers, Inc.

Fennell, R., and Lesser, V.R., (1976). Parallelism in Artificial Intelligence Problem Solving: A case study of Hearsay II. In *Readings in Distributed Artificial Intelligence*, eds. Bond and Gasser. Morgan Kaufmann Publishers, Inc.

Giarrantano, J. (1991) *CLIPS Reference Manual..* NASA Johnson Space Center

Kai Li. 1986 'Shared Virtual Memory on Loosely Coupled Multiprocessors'. Ph.D Thesis, Yale University, Department of Computer Science.

Leao, L.V., and Talukdar, S.N. (1988). COPS: A System For Constructing Multiple Blackboards. In *Readings in Distributed Artificial Intelligence*, eds. Bond and Gasser. Morgan Kaufmann Publishers, Inc.

Lee, K. J., et. al. (1989). *An OOD Paradigm for Flight Simulators, 2nd Edition.* Technical Report of the Software Engineering Institute.

Nii, H.P., Aiello, N., and Rice, J. (1989). Experiments on Cage and Poligon: Measuring the Performance of Parallel Blackboard Systems. In *Distributed Artificial Intelligence, Volume II.* eds. Gasser and Huhns. Morgan Kaufmann Publishers, Inc.

Rossomando, P.J. (1992). The Achievement of Spacecraft Autonomy Through the Thematic Application of Multiple Cooperating Intelligent Agents. *1992 Goddard Conference on Space Applications of Artificial Intelligence*, CP 3141, 87-103.

Zeigler, B. P., (1990). *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic Press Inc.