Real-Time AI Systems: A Definition and An Architecture

Rajendra Dodhiawala, N. S. Sridharan, Peter Raulefs†, Cynthia Pickering
FMC Corporate Technology Center
1205 Coleman Dr., Box 580, Santa Clara, California 95052

Abstract

Speed alone is insufficient for real-time performance. We define real-time performance in terms of speed, responsiveness, timeliness, and graceful adaptation. We claim that all four aspects are essential if a system is to support real-time problem-solving. We also present a distributed knowledge processing architecture based on the blackboard paradigm that addresses all aspects of real-time performance. Primary attention was given to flexibility of behavior without compromising on the efficiency of implementation so that the applicability of the architecture to an application may be experimented with. Performance metrics are crucial for validating real-time performance, and form an integral component of a real-time system. In this paper, we present performance metrics for responsiveness and timeliness at the architecture level.

1. Introduction

Real-time performance issues have only recently begun to attract the attention of the AI community. This is largely spurred by the emergence of real-time applications amenable to AI techniques: manufacturing process-management, in-flight fault diagnosis systems, DARPA's Pilot's Associate and Battle Management programs, and NASA's Mars Rover and Space Station initiatives. Such problems may best be characterized by the highly dynamic environments providing input data as well as one or more interactive users who influence the system's problem-solving processes.

The richness of the applications enable not only testing AI approaches, but generally, incorporating innovative hardware and software approaches. However, all these approaches tend to "overlook" the real-time per formance aspects of the problems in lieu of the "more interesting" non-real-time aspects. We have made real-time issues primary to our research without compromising on other aspects of problem-solving, essential for the complete solution. We firmly believe that a direct solution to real-time issues lies in a performance-oriented, yet flexible knowledge processing software architecture.

The paper introduces, in section 2, a definition of real-time. The real-time architecture is presented in section 3. The performance metrics, defined in terms of the real-time definition, which are used to validate our architecture are discussed in section 4. We follow this with related work described in section 5. Conclusions and future directions are presented in section 6.

2. What do we mean by Real-Time?

Real-time systems have been defined as: "predictably fast enough for use by processes being serviced" ([Marsh and Greenwood, 1986]); "there is a strict time limit by which a system must have produced a response, regardless of the algorithm employed" ([O'Reilly and Cromarty, 1985]); "ability of the system to guarantee a response after a (domain defined) fixed time has elapsed" ([Laffey et al., 1988]); and "[a system] designed to operate with a well-defined measure of reactivity" ([Georgeff, 1988]). These definitions are general and hence and open to interpretation depending on the problem at hand. In fact, Stankovic [1988] presents an excellent analysis of real-time systems which lays out some of the shortcomings of current approaches.

2.1. Four Aspects of Real-Time Performance

While speed is indeed fundamental to real-time performance, speed alone is not real-time. The four aspects of real-time performance are:

- *speed*
- *responsiveness*
- *timeliness*, and
- *graceful adaptation*.

Speed is the rate of execution of tasks. Tasks could refer to problem-solving tasks (large or small) or event-processing tasks.

Responsiveness is the ability of the system to stay alert to incoming events. Since an interactive real-time system is primarily driven by external inputs, the system should recognize that such input is avail able. It may not necessarily process the new event right away; that may depend upon the its criticality relative to other events the system is currently processing.

Timeliness is the ability of the system to react to and meet deadlines, which can be achieved by processing the more critical tasks first. Criticality of tasks may be defined by several factors, either domain dependent or

†Author's current address: AI Center, Intel Corp, P. O. Box 58125, SC9-22, 2250 Mission College Blvd, Santa Clara, CA 95052-8125

domain independent or both. For instance, *urgency* of response to an event is related to the amount of slack time; that is, the difference between the earliest start and latest start for the event to still complete before its deadline. (The deadline for an event is specified by the domain.) Yet another example is the *importance* of an event relative to the other events, strictly a domain issue. (Importance of an event may depend upon the context, hence would be dynamic.)

Graceful adaptation is the ability of the system to reset task priorities according to changes in work load or resource availability. Graceful adaptation allows for a smooth transition across potential perturbations to the system. Without this ability, the system may waste processing resources as well as time by executing less relevant tasks or wait for scarce resources.

It seems imperative that we utilize a knowledge processing approach in order to address all four aspects of real-time performance to solve problems of the type and complexity listed earlier.

3. A Real-Time Architecture

We now present a real-time knowledge processing architecture, RT-1, which conforms to the real-time definition given above and supports evaluating an application in terms of its requirements on speed, responsiveness, timeliness, and graceful adaptation. We will discuss the overall RT-1 architecture first, followed by a description of the component modules and processes.

3.1. Overview of the Architecture

RT-1 is a small-scale, coarse-grained, distributed architecture based on the blackboard paradigm.

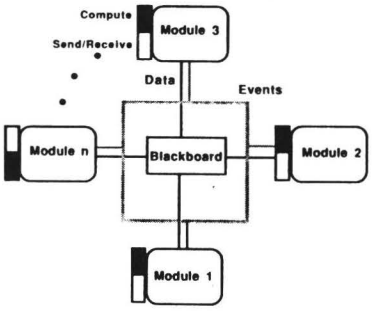

Figure 1: Top Level RT-1 Architecture

The main features of the top level architecture, shown in Figure 1, are:

• It consists of a collection of *reasoning modules* which share a common blackboard dataspace and communicate with each other by signaling events. The reasoning modules run in parallel on independent processors.

• The reasoning modules operate *asynchronously*. Some of the modules in the RT-1 system may be in the communication mode (accessing the blackboard or signaling events) while the others may be in the compute mode (making control decisions or executing domain actions). The asynchronous mode of operation of the modules eliminates the inefficiency associated with synchronous operation in which faster modules have to wait for the slower modules in order to accomplish the synchronization.

• Each reasoning module is *event driven*. Knowledge processing actions in a reasoning module are triggered by events. Events may be signaled by knowledge sources in the reasoning module or by blackboard changes. These events may be made available to all reasoning modules.

• There is an *event directory* which used to route events to their destination modules. Events are signaled by the sender without designating the receivers. The event directory, established at compile time, serves to route an event signaled from a module to the appropriate destination modules interested in processing the event. The advantages of the event directory are the reduction in volume of communication (since events do not have to be broadcast) and that a module receives only those events for which it has the capability and interest to process.

• The blackboard is the shared repository for data common to all reasoning modules. Typically, blackboard data may be used to establish context for knowledge processing actions, generate hypotheses, as well as to control problem-solving within the reasoning module. In a distributed multiprocessing environment, the blackboard is also distributed with each module physically storing only a part of the shared blackboard. The blackboard path facilitates the sharing of information, transparent to the individual reasoning modules.

The RT-1 system is implemented in Genera 7 on the Symbolics Lisp machine, using Flavors and Common Lisp.

3.2. The Reasoning Module

The reasoning module, as shown in Figure 2, consists of three processes linked via blackboard and event paths. The three processes are: the I/O process, the blackboard process, and the reasoning process.

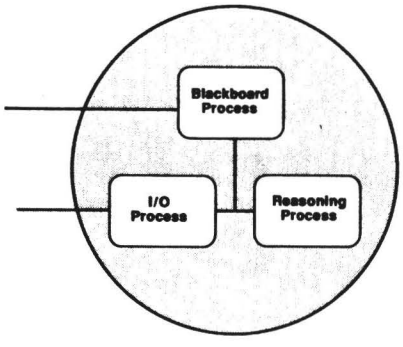

Figure 2: The Reasoning Module Architecture

The I/O process allows for the asynchronous sending and receiving of events to and from other reasoning modules. The blackboard process executes a queue of blackboard demons. The blackboard demons are low level routines initiated as a result of blackboard transactions. The reasoning process performs the knowledge processing actions of the reasoning module.

3.2.1. The I/O Process

The I/O process within a reasoning module receives incoming (external) events from other reasoning modules and sends out events from the reasoning process of its reasoning module to other reasoning modules. The I/O process is capable of queuing the external events on the asynchronous event list while the reasoning process

is in any one of its top level cycle steps. The outgoing events are checked against a global event directory to determine the destinations of the event, and the I/O process then dispatches the events appropriately.

3.2.1. The Blackboard Process

The blackboard is shared by all the reasoning modules. The class definitions of various domain dependent objects reside on the blackboard. At run time, modules may create or delete instances of these objects, or read or modify the slots of the instances. The blackboard objects may have demons associated with these transactions. The demons perform computations (like range checking on data, calculating averages, or time-stamping updates) that are too mundane for the knowledge processing actions in the reasoning process. The blackboard demons execute in the blackboard process and may also signal events which, for instance, may indicate anomalies like "out of range" values. Thus, blackboard demons provide a way to decouple knowledge processing actions (in knowledge sources) from the data validation actions associated with blackboard data. Without the blackboard process, these tasks would have to be performed by the reasoning process, possibly via knowledge sources, resulting in increased overhead in the reasoning process.

The blackboard also holds control information used by the reasoning module and its processes for performing control decisions. Information about goals, plans, their status, current problem-solving strategy, and temporal constraints is stored in the control structures.

3.2.3. The Reasoning Process

The reasoning architecture, shown in Figure 3, is based on the blackboard paradigm. The system is event-driven with knowledge processing actions embodied in knowledge sources and shared data stored on the blackboard.

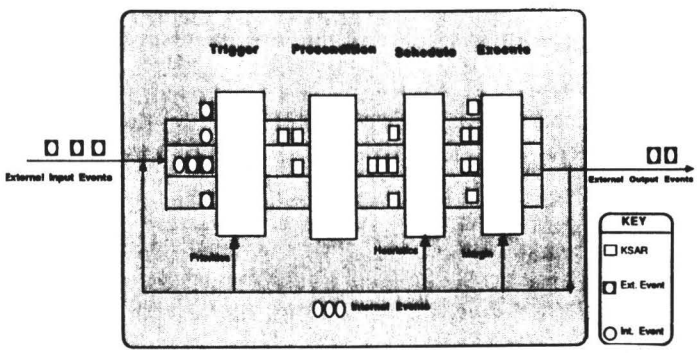

Figure 3: The Reasoning Process

The reasoning process receives external events from the I/O process and the internal events signaled as a result of actions of the knowledge sources. The top level cycle of the reasoning process is made up of four steps: trigger, precondition check, schedule, and execute.

The trigger step is mainly responsible for recognizing the arrival of events and establishing *relevant* response(s) to these changes. An event may trigger knowledge sources which are instantiated as knowledge source activation records (KSARs). See [Hayes-Roth, 1985] for more details on the trigger step.

The precondition check step ascertains for each triggered KSAR whether the information context exists for the execution of the action body of the corresponding knowledge source. This check also permits obviating a KSAR in case the context can no longer be established or the KSAR has simply become redundant or obsolete. Preconditions establish the *readiness* criteria for knowledge source execution.

The schedule step establishes the *appropriateness* of executing the executable knowledge sources at any point in time. It prioritizes the executable KSARs using *control heuristics*. (This is KSAR prioritization, in contrast to event priorities associated with event channels explained below.) The control heuristics defined within a reasoning module establish control regimes for the various goals of the reasoning module and these, in turn, contribute to the goals of the overall system. The set of active heuristics allow focusing the system's resources to achieve the current goal in a timely manner and may be changed dynamically.

The execute step uses the execution margin (described below) to select a subset of the scheduled KSARs and executes, in the same cycle, the action body of the corresponding knowledge sources. The preconditions are checked just prior to execution to reaffirm the information context. If the context is invalid, the failure body of the knowledge source is executed instead. Knowledge sources may signal new events, perform blackboard transactions or even change control parameters that enable decision-making on what actions to prefer next.

The main features of the reasoning process that support real-time performance are the *prioritized event channels*, the explicit *control reasoning* capability, and the regulation of reactivity via *execution margin*. The prioritized event channels allow more critical events to be processed before less critical events. Control reasoning permits opportunistic selection of knowledge sources which best meet the current goal of the reasoning module, and thus, the goal of the overall system. Execution margin allows the system to be completely reactive, completely goal-directed, or any intermediate state that strikes a balance between reactive and goal-directed behavior.

The external events, signaled by other reasoning modules' reasoning processes, are called *asynchronous events*, while the events that are signaled internally within a reasoning process are called *synchronous events*. Associated with each event is a priority. The different event priority levels are determined by the application. The reasoning process is *sliced* into multiple channels, one for each event priority level. These channels extend across the top level cycle of the reasoning process. An event's priority determines the channel to which it is routed. Further processing of the event, as it proceeds through the various steps of the top level cycle, continues on the same channel. The top level cycle is uniformly similar for *all* the event channels, and, as described above, applies to *any* event channel. However, it may be customized for an event channel commensurate with the priority associated with the events on that channel.

The rationale behind the multiple event channels is to pay more attention to events of higher priority. The top level cycle processes events and knowledge sources on the highest event priority level. Only if there is no work to be done on the highest channel will the top level cycle proceed to work on the next lower event

channel. In this way, event and knowledge source processing on the lowest channel is possible only if there are no outstanding events or knowledge sources to be processed on the higher channels. Thus, the reasoning process will be more *responsive* to events in the higher priority channels. However, the increased responsiveness of the higher priority event channels comes at the expense of the reduced responsiveness of the lower priority event channels. From this discussion, it is evident that the reasoning process is a *single* process with a *single* locus of execution. Channel Mix, or allowing more than one channel to be processed in a single top-level cycle, allows for simulated multiprocessing capability within the reasoning process.

Execution margin *amortizes* the control overhead by executing multiple knowledge sources (as compared to single knowledge source execution found in most blackboard systems). It allows trading off responsiveness for timeliness. An execution margin setting that results in selecting only one KSAR from the prioritized agenda and executing its knowledge source allows fast cycle times, thus displaying *responsiveness* to the incoming events. The disadvantage of this setting is that if there are no new events, the response to the goal at hand is being unnecessarily interrupted. However, an execution margin setting that results in selecting all the KSARs on the prioritized agenda and executing the corresponding knowledge sources is goal-directed, displaying *timeliness* to system goals. The disadvantage of this setting is lack of responsiveness to incoming, possibly critical events. The spectrum of values of execution margin between these two extremes establishes a compromise in responsiveness and timeliness. The output of the execute step is a set of events created by the executed knowledge sources. Realize that an event signaled from a knowledge source may have a different priority assignment than the priority of the event that signaled the knowledge source. Hence, new event priorities may change dynamically via knowledge source execution.

The reasoning process offers considerable flexibility in experimenting with the various features. In fact, the strength of the architecture is just that: it is malleable enough to allow adjusting the features of the system for a variety of applications.

4. Performance Metrics

The RT-1 system is well instrumented. We measure various aspects of the performance of the system in a separate process, not discussed above. The probes in the system are used to compute various performance metrics and indicators. Currently, these metrics and measures are defined at the architecture level. They permit experimentation with the features of the system and primarily help to analyze and validate the architecture. The first phase of our performance metering and experiments addressed different aspects of responsiveness and timeliness. Evaluating the architecture for graceful adaptation is not complete at this time and will be reported later.

4.1. Real-Time Performance Measures

We describe the real-time performance of the RT-1 system in terms of *measures, metrics,* and *probes.* A *performance measure* is a quantity characterizing a particular aspect of real-time performance. Ranges of values for performance measures are generally unbounded. A *performance metric* is a normalized measure so that there are upper and lower limits for the values of a metric, usually related to best and worst case performance. A *performance probe* allows the RT-1 system to collect performance data throughout a run.

Performance measures allow comparisons between the system's performance for the different settings of the system parameters, such as execution margin and control heuristics. Performance metrics provide the basis for rating what has been achieved with regard to optimal and worst performance and for making further judgements based on those ratings. Performance measures and metrics are computed from data collected by the probes.

4.2. Responsiveness Measures

The *responsiveness* of a system is inversely related to the *channel latency* or the total time it takes to notice a new event, and to begin composing a response for the event. The *response task* for an event *e* includes the set of KSARs triggered by event *e* and eventually executed. The *response latency* is the time between the signaling of an event until its response task completes. Response latency is computed from time-stamps that the RT-1 system attaches to events. The five probes for an event *e* are:

1. *sending-time(e)* is the time at which *e* is signaled.
2. *receiving-time(e)* is the time at which *e* is received by a reasoning process and placed on one of its event channels.
3. *begin-ack-time(e)* is the time at which *e* is acknowledged by a reasoning process; that is, the time at which the reasoning process starts triggering KSARs off the event.
4. *end-scheduling-time[cycle#(e)]* is the time at which the scheduling step of the top level cycle for the cycle numbered cycle#(*e*) ends. Here, cycle#(*e*) is the cycle in which *e* was acknowledged.
5. *event-response-time(e)* is the time at which execution of the last KSAR in the response task of *e* terminates.

From these time-stamps, we can compute the following latencies:

1. The *communication latency* for an event *e* is the delay between the time *e* is signaled and the time *e* is received by a reasoning process. (See R1 in Figure 4.)
2. The *channel latency* for an event *e* is the time for which *e* stays on an event channel of a reasoning process, starting at the time it is received by the reasoning process, and ending at the time the reasoning process starts triggering KSARs from the event. (See R2 in Figure 4.)
3. The *TPS latency* for an event *e* is the time the system spends on triggering(T) KSARs from *e*, checking preconditions(P) of the triggered KSARs, and scheduling(S) the activated KSARs, all in the cycle where KSARs are first triggered off event e. The TPS latency is the time delay caused by activating and scheduling the response task for event *e* in the cycle where event *e* is noticed or acknowledged. It does not include delays for checking preconditions and re-scheduling KSARs that remain in the system after the first cycle of noticing event *e*. (See R3 in Figure 4.)
4. The *execution latency* of an event *e* is the delay between the time when the KSARs in the response task

of *e* are first scheduled and the time at which the last KSAR responding to *e* terminates execution. (See R4 in Figure 4.)

The *total response latency* for an event *e* is the sum of the above four latencies. It reflects the time delay between occurrence of *e* and completion of the response to e. (See R5 in Figure 4.)

The average channel latency is the indicator of the responsiveness of a channel. If the average channel latency is low, the system is extremely responsive to the events on that channel. It is evident that the channel latency deteriorates as one goes from the highest priority channel to the lowest priority channel. Also, the channel latency is primarily influenced by the following features, taken one at time:

- The channel latency increases with increasing cycle time of the same or lower channels.
- The channel latency increases with increasing values of the execution margin setting of the same or lower channel.
- The channel latency is affected by channel mix. The channel latency of the higher channel increases while that of the lower channel decreases with channel mix turned on.
- The channel latency is higher if the granularity of knowledge source is large on the same or lower channels; lower otherwise.

A good metric for responsiveness is

channel-latency(*e*)/total-response-latency(*e*).

In general, small cycle times improves responsiveness.

4.3. Timeliness Measures

We make two assumptions about event deadlines:

- Each synchronous and asynchronous event received by a reasoning process has a required *deadline* for completing its response task. The deadline is specified as a delay after the sending time of the event.
- For each event, it is possible to recognize when a response for that event is completed.

We define Event Response Timeliness (ERT) as:

ERT(*e*) := response-time(*e*) - deadline-time(*e*)

ERT is zero if the response is on time, negative if early and positive if late.

ERT gives a measure of lateness, and can be used to measure the how well the system did on an individual event basis. However, a more global perspective is desirable, possibly over sets of significant events, or *event windows*. Event windows may either be based on event receiving times or event response times. Very simply, the complete processing cycle is an event window. Averaging over ERT values within a window of both early and late responses causes negative and positive values to cancel each other, and destroys information about the distribution of response timeliness. We propose L-ERT and LW-ERT measures to include only late events.

Late ERT (L-ERT) averages ERTs of those events in the event window *w* which are late, as given in equation T1 in Figure 4.

Since not all events are equally important, Late Weighted ERT (LW-ERT) takes a weighted average of ERTs of late events in a given event window. Equation for LW-ERT is given in T2, Figure 4.

Both L-ERT(*w*) and LW-ERT(*w*) are unbounded non-negative numbers. Corresponding averages may be computed for early responses.

One possible lateness-factor may be the *late-importance-mass-ratio* (late-imass-ratio) of an event, defined as equation T3 in Figure 4, where importance(*k*) is the importance rating of the KSAR *k* by a domain specific control heuristic, and response-task(*e*) the set of KSARs that were triggered by *e* and ultimately executed. Using late-imass-ratio as the lateness factor in LW-ERT weights events by the portion of the response that missed the deadline.

The ratio LW-ERT(*w*)/L-ERT(*w*) is a metric defined only for windows where some events have late responses. LW-ERT(*w*) $\leq$ L-ERT(*w*) since late-imass-ratio ranges between [0,1], and the metric tends towards 0 as less mass is completed late. Thus, it indicates whether the system (and the particular control regime used during *w*) prefers more important events.

5. Related Work

There is ample literature on real-time systems in general. Real-time AI has not received widespread attention until recently, particularly when conventional approaches fall short of handling the complexity of today's sophisticated applications. Most of the issues that are primary in our research are succinctly expressed by Stankovic, [1988]. The survey by [Laffey *et al.*, 1988] is a good starting point of getting acquainted with the state of real-time AI. The work in process control ([Raulefs and Thorndyke, 1987, Raulefs *et al.*, 1987]) was the primary motivation for our interest in real-time problem-solving. [Dodhiawala, *et al.*, 1989] is an expanded version of this paper, reporting some of the motivations behind the RT-1 architecture, and results of early experiments.

R1: communication-latency(e) = receiving-time(e) - sending-time(e)

R2: channel-latency(e) = begin-ack-time(e) - receiving-time(e)

R3: TPS-latency(e) = end-scheduling-time[cycle#(e)] - begin-ack-time(e)

R4: execution-latency(e) = event-response-time(e) - end-scheduling-time[cycle#(e)]

R5: total-response-latency(e) = communication-latency(e) + channel-latency(e) + TPS-latency(e) + execution-latency(e)

T1: $\text{L-ERT(w)} = \dfrac{\{\sum \text{ERT(e)} \mid \text{e in w \& response of e is late}\}}{\text{number of events in w}}$

T2: $\text{LW-ERT(w)} = \dfrac{\{\sum[\text{ERT(e)} \times \text{lateness-factor(e)}] \mid \text{e in w \& response of e is late}\}}{\text{number of events in w}}$

T3: $\text{late-imass-ratio(e)} = \dfrac{\{\sum \text{importance(k)} \mid \text{k in response task of e \& k did not complete before the deadline of e}\}}{\{\text{importance(k)} \mid \text{k belongs to response task(e)}\}}$

Figure 4: Responsiveness and Timeliness measures

Current work on Guardian, [Hewett and Hayes-Roth, 1988], addresses some of the issues in asynchronous communication in BB1, leaving the main inference engine almost intact. Their primary goal is to deliver sensor data in a timely manner, with the reasoning process capable of setting sensor sampling rates and fusion commands.

Real-time planning as addressed by Durfee and Lesser ([Durfee and Lesser, 1987]), raises issues of appropriate level of planning mainly because plans may change due to unforseen implications of plan actions or to new information from the environment that violates plan assumptions. The issues of approximate reasoning in [Lesser *et al.*, 1988] have been part of our research also. In fact, LW-ERT based on the late-imass-ratio as the lateness factor gives an indication of how the system did in terms of partial results within the event's deadline. In RT-1, it may be used in an anticipatory fashion to determine how good an answer may be made available in the given time.

PRS, [Georgeff, 1988], utilizes a variant of the blackboard architecture and primarily addresses issues of reactive responses and meta-level reasoning comparable to our event priority channels and control reasoning. However, PRS lacks the flexibility we offer in RT-1. One advantage of PRS is its Knowledge Area formalism. It allows sequencing of knowledge sources and already contains a lot of context that we have to establish in RT-1 knowledge sources at trigger time.

Work on distributed GBB, [Corkill, 1988], is mainly at the blackboard level, not at the control architecture and has come out of work on distributed problem-solving, planning and real-time performance.

6. Future Directions

We believe that we have made significant progress toward understanding real-time issues and performance criteria. However, our work is only a step in this direction. Several major issues still remain. Some of these relate to the capability of the RT-1 architecture, while others are general real-time issues. The next set of research activity involving the architecture includes extension for graceful adaptation, more intelligent schedulers, control overhead issues, interruptibility of knowledge sources, and global coherence in a distributed setting. General real-time issues include guaranteeing response, predicting response times and solution quality, incremental algorithms, and anticipatory processing.

Acknowledgments

This work has benefited greatly from the advice and criticisms offered by Perry Thorndyke and Barbara Hayes-Roth. We also thank Tom Hester for his support, Bill Murray for helping out with the implementation, and Charles Key for reviewing earlier drafts of this paper.

References

[Corkill, 1988] D. D. Corkill. Design Alternatives for Parallel and Distributed Blackboard Systems. In *Proc. of the Second blackboard Workshop*, pages 89-106. AAAI, St. Paul, MN, August, 1988.

[Dodhiawala *et al.*, 1989] R. T. Dodhiawala, N. S. Sridharan and C. Pickering. "A Real-Time Blackboard Architecture. To Appear in *Blackboard Architecture and Applications: Current Trends*, Eds. V. Jagannathan, R. T. Dodhiawala and L. Baum, Academic Press, 1989.

[Durfee and Lesser, 1987] E. H. Durfee and V. R. Lesser. Incremental Planning to Control a Blackboard-Based Problem Solver. In *JPL Planning Conference*, pages 91-99. JPL, California, 1987.

[Georgeff, 1988] M. P. Georgeff. An Embedded Reasoning and Planning System. In *Advanced Proc. of the Rochester Planning Workshop*, pages 79-101. University of Rochester Computer Science Dept., Rochester, NY, October, 1988.

[Hayes-Roth, 1985] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence Journal* 26:251-321, 1985.

[Hewett and Hayes-Roth, 1988] M. Hewett and Hayes-Roth. Real-Time I/O in Knowledge-Based Systems. In *Proc. of the Second Blackboard Work shop*, pages 107-118. AAAI, St. Paul, MN, August, 1988.

[Laffey *et al.*, 1988] T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao, and J. Y. Read. Real-Time Knowledge-Based Systems. *AI Magazine* 9(1):27-45, Spring, 1988.

[Lesser *et al.*, 1988] V. R. Lesser, J. Pavlin, and E. H. Durfee. Approximate Processing in Real-Time Problem-Solving. *AI Magazine* 9(1):49-61, Spring, 1988.

[Marsh and Greenwood, 1986] J. Marsh and J. Greenwood. Real-Time AI: Software Architecture Issues. In *National Aerospace and Electronics Conference*, pages 67-77. IEEE, Washington, D. C., 1986.

[O'Reilly and Cromarty, 1985] C. A. O'Reilly and A. S. Cromarty. 'Fast' is not 'Real-Time' in Designing Effective Real-Time AI Systems. In *Applications of Artificial Intelligence II*, pages 249-257. Int. Soc. of Optical Engineering, Bellingham, Washington, 1985.

[Raulefs *et al.*, 1987] P. Raulefs and P. W. Thorndyke. An Architecture for Heuristic Control of Real-Time Processes. In *Proc. NASA/JPL Workshop on Space and Telerobotics*. NASA/JPL, January, 1987.

[Raulefs, *et al.*, 1987] P. Raulefs, B. D'Ambrosio, M. R. Fehling and S. Forrest. Real-Time Process Management for Materials Compositions in Chemical Manufacturing. In *IEEE Conference on Applications of Artificial Intelligence*, pages 120-125. IEEE, 1987.

[Stankovic, 1988] J. A. Stankovic. A Serious Problem for Next-Generation Systems. *IEEE Computer* 10(21):10-19, 1988.