A Real-Time Knowledge Processing Executive for Army Rotorcraft Applications

Dan R. Ballard
David Nielsen
Reticular Systems, Inc.

Abstract
Efforts such as the Army's Rotorcraft Pilot's Associate (RPA) Advanced Technology Transition Demonstration (ATTD) Program have requirements for real-time execution of knowledge-based systems. However, existing computer hardware platforms and software systems (languages, tools, shells, etc.) are not adequate for applications requiring real-time knowledge-based processing. This is not surprising since knowledge-based systems have historically been developed with non-real-time applications as their primary focus (e.g., expert medical consultants, off-line diagnostics, advisors, etc.). Further, these applications generally were not concerned with run-time efficiency, optimal use of computer resources, embedded system requirements (size, weight and power) and other considerations important in addressing real-time processing. The fundamental design concepts necessary for real-time performance were not considered in the design of the languages, shells and tools used for knowledge-based processing.

This paper describes research in the development of a real-time knowledge-based system execution environment suitable for use in next-generation Army helicopters. The paper first describes the unique problems which must be solved in real-time knowledge processing applications. We then describe an overall processing architecture for achieving the requisite real-time performance. The execution environment is being specifically designed for embedded applications such as the mission equipment package of the RAH-66 Comanche helicopter. The environment will provide the requisite software execution facilities which heretofore have not been available in an integrated embedded execution environment. Further, this execution environment will provide the necessary control features required to manage real-time problem solving.

What is Real-Time AI Processing?
Traditionally, computer science has defined a real-time system as a system which must not only provide the *correct* answer but provide it within strict time constraints. As we shall see, this definition is not adequate for AI processing systems. AI researchers have no generally accepted definition of *real-time AI*. Many have assumed that faster execution is an acceptable method for attaining real-time performance. To that end, previous approaches to real-time AI have concentrated on speed-up of the process. This speed-up has been accomplished in a number of ways. Solutions tried include:

- Separation of development and run-time environments.
- Use compiled rather than interpreted code.
- Use a faster language, e.g., C rather than LISP.
- Use special purpose hardware, e.g., associative processors, LISP machines and parallel architectures.

For example Marsh states that "A system exhibits real time behavior if it is predictably fast enough for use by the process being serviced."[1] Certainly, a real-time AI processor must provide timely response, i.e., it must be able to provide an answer when needed. Data will be received from the external environment at a certain rate, and the system must make the best decisions and give the best possible response within the allotted processing time [2].

O'Reilly and Cromarty provide a slightly more rigid definition. They require that "there is a strict time limit by which the system must have produced a response, regardless of the algorithm employed."[3] However, providing a response within a strict time limit is difficult. This is because of the nature of AI processing and the nature of the problems solved using AI techniques. AI systems are, by nature, adaptable to both their external and processing environments. The complexity of these systems prevents *a priori* knowledge of their behavior. Thus it is impossible to precalculate all possible combinations of tasks which may occur. Therefore, we cannot be assured that the system will arrive at the one *correct* answer within a certain time constraint.

Guaranteed response time is essential in hard real-time systems. The system must be able to assess the time it has available to achieve a goal, provide different levels of solution that may range from reflex-type reactions to in-depth reflective inferencing, and assess goal priorities and timings in order to make the appropriate resource allocations.

A real-time AI processing system must he able to operate and manage limited available computational resources (i.e., computing power, processing time, memory space). It must have an efficient means of focusing attention on relevant portions of the domain without ignoring on-going processes. The system will have to decide what actions to take based on multiple (and not necessarily consistent) objectives [2).

A real-time system must provide the best possible answer in the time allowed for processing. For many real-time applications, a variety of tasks will exist in which the amount of time available to make a decision does not always allow time for a considered decision. In some cases, it is desirable to make a hasty decision in limited time if sufficient time is not available for making a more considered decision [4].

A real-time AI system will use approaches that allow programs to meet deadlines (i.e., real-time constraints). Conventional approaches assume that a task's priorities and resource needs (including time) are known *a priori* and are unrelated to other tasks so that the operating system or executive can schedule tasks based on their individual characteristics. If more tasks exist than the system can process, then the system simply ignores the task (usually based on priority). The basic notion required for intelligent real-time knowledge-based systems is that time is an integral component to planning an execution. Recognizing imminent deadlines in solutions should cause the system to revert to approximate reasoning methods [5].

AI based tasks are interdependent because they search different parts of the solution space to solve related subproblems. Because the solution space is too large to search exhaustively, search is heuristic and only a small number of potential tasks are performed. A problem solver may be able to make estimates about the amount of time required to solve a problem. However, it cannot precisely determine the time needed because the information that influences heuristic decisions might change as the search progresses. Moreover, solving some subproblems can affect the importance and time required to solve related subproblems.

A focusing mechanism constrains search (in the inference process) for knowledge that is relevant to the current state of the problem and is thus eligible for execution. This is important because many inferencing mechanisms implement search algorithms which, if not constrained, result in combinatorial explosion. Focusing requires explicitly representing control knowledge and making this knowledge available to the

inference process. Control knowledge is used to guide the inference process and focus it on the relevant and important portions of the KB. Control knowledge is used to resolve trade-offs (such as immediacy vs. accuracy or degree of certainty vs. level of abstraction) and prioritize sub-problems.

Given the nature of AI processing, we define a real-time AI system as a system which can guarantee an *acceptable* solution within the available time constraints. The acceptability of the solution and the available time constraints are determined by the system designer.

Issues In Real-Time AI System Design

Since, we are interested in obtaining an appropriate answer under time constraints, it is important to consider the performance limitations placed on AI systems by the underlying algorithms and data structures. The goal of our research has been to address and understand the underlying problems with search algorithms, pattern matching algorithms, inferencing techniques, control methods and knowledge representation schema used in real-time systems. A real-time system must be designed with careful consideration to the performance attainable using these underlying algorithms and representation schema.

Search Algorithms

Search algorithms are ubiquitous to AI systems. Many times, search algorithms are included in a system design without consideration of how their performance impacts overall knowledge processing performance. Analysis by O'Reilly and Cromarty [3] has shown that search time in a production system which uses forward chaining increases exponentially with an increase in the search space. The number of rules that will fire increases exponentially with the depth of the inference tree.

They also showed that in backward chaining (whether using breadth-first or depth-first search), every increment in the depth of the tree gives an exponential increase in the number of tree nodes and a combinatorial increase in the number of paths to search. The greater the branching factor of a node, the quicker the exponential increase.

It should be noted that even though a search algorithm's performance degrades exponentially, it is still possible to place bounds on the search time. However, a worst case analysis may result in search times that are unacceptable for many applications. Obviously more powerful methods of handling search are required. Recent research by Korf provides an example of how improved search algorithms can improve overall system performance and enhance the overall "intelligent" problem solving of knowledge based systems. Korf has developed search algorithms which provide the ability to commit to an action almost instantaneously, but allow the quality of that decision to improve as long as time is available. Once a deadline is reachable, the best decision arrived at is executed. For instance his Real-Time A* (RTA*) algorithm allows backtracking while still guaranteeing a solution and making locally optimal decisions [6-10].

Pattern Matching Algorithms

The Rete algorithm is the most common algorithm for use in production system pattern matching and is found in most systems including CLIPS, ART and OPS5 [11]. Rete was initially developed for use in non-real-time systems and exhibits unpredictable response times. Haley has addressed problems with the Rete algorithm when used in real time. His major concern is needing proof rather than accepting the real-time performance of a less-than-exhaustive set of test cases. According to Haley, the Rete algorithm is not well suited to the formal analysis required to assure real-time performance [12]. His analysis shows that one cannot *a priori* calculate the cost (that is the number of joins, where a join is an instantiation of a variable across patterns) of adding an assertion to the working memory using the Rete algorithm. Instantiating a join results in unpredictable response times. There are a number of possible solutions to the shortcomings of the Rete algorithm. Haley presents several methods for bounding the cost of

matching a rule including join matching limitations, pattern instantiation restrictions, relation instance restrictions and cardinality restrictions.

There exists the possibility that these restrictions might compromise some of the power and flexibility of the production system approach to problem solving. For non-developmental (i.e., embedded) environments, these restrictions may be acceptable given that they can help guarantee real-time performance.

Knowledge Representation

Knowledge representation issues concern both representation of the knowledge base (KB) and the incoming data. The mechanisms used for knowledge representation significantly influence the performance of a knowledge processing system. Frame languages use an *instance* relation to allow properties and default values to be inherited from generic type frames and retrieve and process all instances of a given type at run time. By adding a second relation, *is-a*, and using it to organize type frames in an organizational hierarchy, frame languages can support object-oriented programming.

The inheritance network forms a tree with a single root. By *inheritance*, we mean that each class can have at most one superclass and only two relations are defined, *is-a* and *instance*. Retrieving a value from a slot consists of following a simple linear list. Worst case times can easily be calculated for retrieving and storing values in the hierarchical tree (O(*d*)), where *d* is the depth of the inheritance tree. Thus, frame languages using only simple inheritance are suitable for keeping tight bounds on response times in real-lime problems [4].

Newer and more powerful languages (e.g., the latest version of C++) allow defining new relations between frames. Objects can have more than one superclass (i.e., multiple or mixed inheritance). Although multiple inheritance allows the user to gain additional expressiveness, it brings a new class of problems that must be solved. An inheritance network, in effect, becomes an arbitrary directed graph. Retrieving a value from a slot now involves some type of search (depth first or breadth first). These types of strategies are not be suitable for real-time applications because of their exponential nature. Therefore the knowledge representation methods used for an embedded real-time system may be coil-strained to the simple frame-based representations.

Knowledge Organization

The organization of knowledge within the real-time processing environment also significantly affects real-time performance. Knowledge must be organized in such a way to make knowledge access more efficient. A number of researchers have suggested knowledge chunking as an effective way of organizing knowledge for real-time applications. The basic idea is to organize knowledge in modules or chunks. In the course of problem solving, if no production rule can be found that solves a problem, the system can reference (a chunk of) deep knowledge that may suggest a multi-step sequence of productions to solve the problem. This sequence is then added to the KB so it will be readily available as a single macro when needed again. Modular chunks can exist in either an on or off state. By associating knowledge with attributes, the inference engine can be invoked by referencing specific attributes [13].

Active knowledge (i.e., rules) can be encapsulated in modules whose properties (time and resource requirements, synchronization requirements, conflicts, etc.) can be expressed, reasoned about, and controlled. These modules are organized into generalized scripts we term Skeletal Processing Structures (SPS) that correspond to pre-formulated processing plans designed to provide responses to the system's processing requirements. At run-time, meta-knowledge operates on these scripts and creates specialized instances to satisfy specific constraints and then executes them.

SPSs allow knowledge about the control aspects and processing resource requirements of the system to be encoded so that the system can intelligently plan its own activity to meet deadlines and make the most effective use of its resources. An SPS is composed of elements representing the computation required to perform some primitive function or sub-functional step. Associated with each element is a time function (or a set of time functions) that estimates the time required to execute this SPS element in its current state, the implicit knowledge about that element's relative position in the structure and meta-knowledge about the effect of this element's invocation on other elements.

An SPS element may support a set of alternative computing strategies to perform its function. These alternative strategies trade off time (or other resources) for solution quality, allowing the system to provide hasty but adequate answers when severely time-constrained, but also to provide high-quality solutions under less stressful circumstances.

Each SP structure has a local controller that accepts external input in the form of time and resource constraints. The local controller composes (elaborates) an instance of the structure by selecting from alternative processing strategies that will perform the required function within the imposed constraints.

Nonmonotonicity and Truth Maintenance

The RTAI environment is dynamic and nonmonotonic. Data received and conclusions reached by the system may have to be changed when new information is received that alters the support for a previous conclusion. Sometimes, data merely gets out of date. The system must take care of maintaining the consistency and integrity of the KB. Using a Truth Maintenance System (TMS), revision of belief is handled by finding the working memory element (WME) that must be adjusted and adjusting this and all WMEs supported by it. For example, if the evidence supporting a conclusion is adjusted (say from being believed to being disbelieved), this evidence is withdrawn together with the conclusions derived from it and the other conclusions derived from the first conclusions and so on along the chain [2].

Continuous Operation

Real-time systems run continuously. For continuous operation, there must be a mechanism to clean up old memory elements, either by deleting them or by archiving them for later retrieval as necessary. In addition, the system must have an efficient method for continuously allocating. deallocating and garbage collection of unused memory. Memory management overhead must not degrade the performance of the system. WMEs that have served their purpose must be removed. In continuous operation, accumulation of old useless data in the working memory not only creates a memory space problem but can also instantiate the wrong productions in a data driven production engine. Removal of multiple WMEs must be done with care such that the system does not unintentionally trigger rules which might be satisfied when only a partial set of working memory elements have been removed. For example, the ability of a rule to fire may depend not only on the presence of some elements, but also on the absence of others. Continuous operation also implies that there may be times when there are no rules activated and wailing to fire. The inference engine must not terminate because no rule is eligible to fire.

Asynchronicity

External events such as sensor inputs occur asynchronously. In addition, events may have different levels of importance. Therefore, the system may have to reschedule processing of less important tasks in order to handle priority interrupts. To provide this capability, the real-time processing environment must accommodate interrupts on two levels. At a low level, interrupts are used to invoke special procedure (e.g., filter input data). At a higher level, interrupts must provide for the re-direction of control as the result of the asynchronous external event. This differs from normal interrupt processing in that the control path in knowledge-based systems cannot be represented simply as a new program counter value, and further, the consistency of the state context (which may be visible to both interrupted and interrupting control

paths) must be maintained. Priorities assigned to a task may be context dependent based on the total, current operational state of the system.

Temporal Reasoning

A real-time system must have the ability to reason about past, present and future events as well as the sequence in which events happen. Time may serve as a gating factor for computational resources and also acts as a description for certain data and knowledge characteristics. The implication is that time dependent variables must be represented in the KB and that inferencing techniques exploit these properties to find rules more efficiently.

Temporal reasoning requires a logic for describing and relationships about events and time. Allen suggests thirteen possible relationships (X before Y, X equal Y, X during Y, etc.) which must be considered in the inferencing process [14, 15]. Gallon extended Allen's work to cover continuous processes occurring throughout time [16]. Other work by McDermott [17] and Dean [18-20] is also relevant to temporal reasoning.

A Real-Time AI Processing Environment

The previous paragraphs have addressed some of the critical issues impacting the design of real-time AI systems. The following paragraphs describe the architecture of a real-time execution environment currently under development at Reticular Systems, Inc. This environment was purposefully designed to handle the unique problems encountered in real-time AI processing. The environment provides the software support tools, libraries and infrastructure required for executing teal-time AI applications in an embedded environment.

Figure 1 illustrates the conceptual organization of the execution environment. The execution environment is modeled as a seven layer stack. Each layer of the stack provides certain services to the execution environment. The bottom Physical Layer represents the physical hardware that executes the real-time system. Minimum hardware resources include a real-time clock and interrupt recognition and handling hardware.

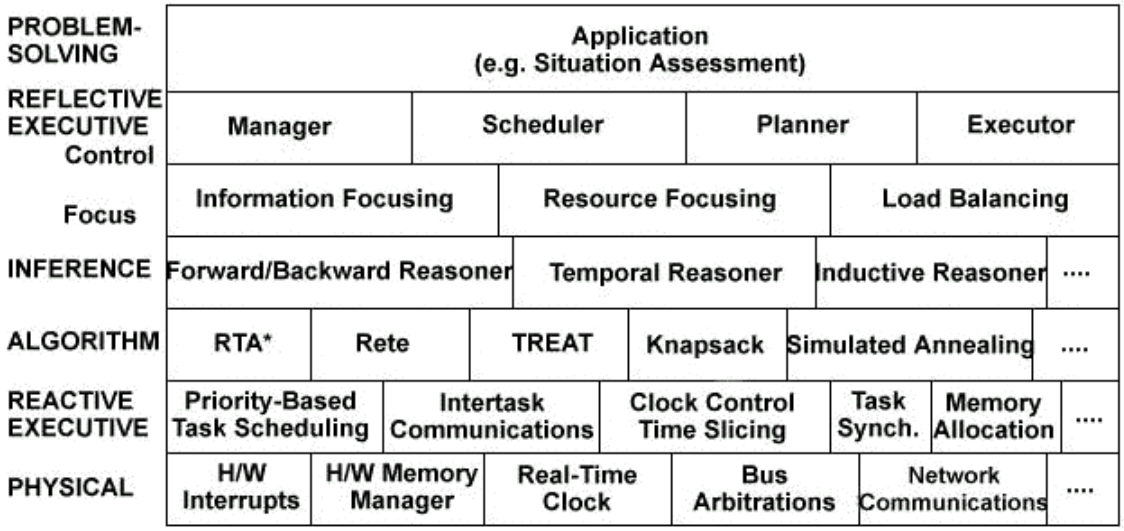

Figure 1. *Real-Time Knowledge Processing Environment*

The next layer of the environment is the Reactive Executive Layer. The Reactive Executive Layer provides many of the services that a traditional real-time executive provides. Typical services include memory management, task scheduling and task synchronization as well as interrupt service.

The Algorithmic Support layer provides a library of high-performance algorithms which are optimized for real-time execution and are available for use by upper layers of the execution environment. Algorithms include RTA*, Rete and TREAT match algorithms and various dynamic programming algorithms (e.g., simulated annealing) which are required for AI problem solving.

The next layer in the stack is in many ways the most important layer. The Reflexive Executive layer provides the interface between the inference mechanism and the application program(s). The Reflexive Executive Layer is responsible for controlling the overall problem-solving process and provides for both focus of *controlling* the overall problem-solving process and provides for both *focus of attention* and control of system operation. The following paragraphs describe these two functions in more detail.

Focus of attention

When a significant event occurs, a real-time system must be able to focus its resources on the important goals. This may mean (a) employing new knowledge, (b) modify the set of sensors currently being used and/or changing the rate at which data is being analyzed.

The critical constraint is to act within a time budget. In recognizing time constraints, a system must recognize goals as well as alternative paths to reach them. Thus dynamic circumstances require the system to focus on immediate outcomes and assess the outcomes in light of the over-riding goals. To modify the processing steps, both problem solving goals and the control strategies must be explicit and modifiable. The system must focus on sub-task or sub-goal evaluation and select the correct inferencing procedures [13].

Hayes-Roth suggest that there are three kinds of focus operations necessary for use in real-time intelligent systems [21-23]. These are *information-focusing* operations, *resource-focusing* operations and *load-balancing* operations. Information-focusing operations are triggered by changes in the systems meta-level control plan and sends focus instructions to the sensor input subsystem to discriminate incoming sensor data. Resource-focusing operations are also triggered by changes in the system's meta-level control plan and "modulates" the overall input data rate in anticipation of changing resource requirements. Load-balancing operations are triggered by overflow or underflow of input buffers and also modulate the overall input data rate but do so in response to unanticipated changes in resource demands. With these operations, a system can focus its attention in accordance with the current meta-level control plans, goals and available resources. The system is thus able to protect itself from being swamped by non-critical inputs. However, the system also remains sensitive to exceptional events outside its current focus of attention.

The first goal of the real-time knowledge architecture is to provide guaranteed responses times for critical events. This implies, first, the ability to identify critical events when they occur; second, the ability to establish deadlines for processing completion (knowing when the answer or response is required); and third, the ability to re-direct appropriate resources to accomplish the needed processing, interrupting (in a recoverable way) current processing, if necessary. In addition, this implies the ability to predict (or control) the execution time of knowledge-based modules. A problem solver's control component cannot and should not reason about individual tasks but instead should reason about how groups of tasks lead to acceptable overall solutions.

Control

While faster hardware and efficient and predictable use of resources by the underlying operating system can improve the real-time processing of individual tasks, a problem solver needs real-time control mechanisms for dealing with the larger grained issues of solving complex problems involving hundred of interdependent tasks. The control mechanism must adoptively generate the most acceptable solutions which meet deadlines and the users' needs when they cannot find optimal solutions in time.

The components of the control mechanism include a *control-manager*, a *control-scheduler*, a *control-executor* and a *control-planner*. The control-manager uses recent events to identify and rate executable reasoning operations. The control-scheduler determines which of the identified executable operations should be executed and when to execute them, based on their ratings. The control-executor executes each operation. The control planner generates a temporally organized pattern of control decisions, each of which describes a class of operations that the environment needs to perform over some period of time. Multiple plans for performing concurrent tasks may coexist in the control plan. This model for the control mechanism is motivated by the intelligent agent architecture of [21-23].

The top layer of the execution environment is the Problem-Solving or Application layer. The user's AI-based application program is situated at this layer and uses all of the resources in the lower layers. If SPSs are used in constructing the application program, then the reflective executive layer can control the scheduling and execution of each SPS.

Summary

We have described our research and development efforts in defining a software environment suitable for execution of real-time embedded AI programs. The execution environment provides the software tools and infrastructure required for execution of knowledge-based software. The environment provides both high-level and low-level executive control, scheduling and management functions. The real-time execution environment provides:

- an architecture that allows timely, efficient, and continuous operation, good knowledge about the domain, a focusing mechanism and working memory that allows the representation of alternative and competing hypotheses. The KB includes both domain knowledge, working memory, and control knowledge to explicitly represent control strategies that guide the inference process.
- an inference engine that includes an intelligent scheduler and interpreter, a mechanism for handling uncertainty and maintaining KB integrity, a temporal reasoning mechanism, an interrupt handler with the ability to receive asynchronous inputs and memory management, garbage collection and archiving facilities.

Bibliography

[1] J. Marsh and J. Greenwood, "Real-time AI: Software architecture issues," in Proceedings of IEEE 1986 National Aerospace and Electronics Conference, Washington, D.C., 1986, pp. 27 - 77.
[2] A. C. Diaz, "An Overview of Real-time Expert Systems," National Research Council Canada - Research Report: NRC No. 31759, April 1990.
[3] D. A. O'Reilly and A. Cromarty, ""Fast" is not real-time," in Applications of Artificial Intelligence II. International Society of Optical Engineering, 1985, pp. 249 - 257.
[4] T. Laffey, P. Cox, J. Schmidt, S. Kao and J. Read, "Real-time knowledge-based systems," The AI Magazine, vol. 9, no. 1, pp. 27-45, 1988.
[5] V. R. Lesser, J. Pavilin and E. Durfee, "Approximate processing in real-time problem solving," AI Magazine, vol. 9, no. 1 (Spring), pp. 49 - 61, 1988.
[6] R. E. Korf, "Real-time heuristic search: First results," in Proceedings of Proceedings of the National Conference on Artificial Intelligence, Seattle, WA, 1987.

[7] R. E. Korf, "Real-time heuristic search: New results," in Proceedings of National Conference on Artificial Intelligence, St. Paul, MN, 1988, pp. 139 -144.
[8] R. E. Korf, "Depth-limited search for real-time problem solving," Journal of Real-Time Systems, vol. 2, no. 112, pp. 7 - 24, 1990.
[9] R. E. Korf, "Real-time heuristic search," Artificial Intelligence, vol. 42, no. 213, pp. 189 - 211, 1990.
[10] R. E. Korf, "Multi-player alpha-beta pruning," Artificial Intelligence, vol. 48, pp. 99 - 111, 1991.
[11] C. L. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem," Artificial Intelligence, vol. 19, pp. 17 - 37, 1982.
[12] P. Haley, "Real-time for RETE," in Proceedings of ROBEXS '87: The Third Annual Workshop on Robotics and Expert Systems, Research Triangle Park, N.C., 1987.
[13] G. V. Ricketts, "How to do more in less time," AI Expert, January 1988.
[14] J. Allen, "Maintaining knowledge about temporal constraints," Communications of the ACM, vol. 32, no. 3, pp. 832-843, 1983.
[15] J. F. Allen, "Towards a general theory of action and time," Artificial Intelligence, vol. 23, pp. 123 -154, 1984.
[16] A. Galton, "A critical examination of Allen's theory of action and time," Artificial Intelligence, vol. 42, no. 2-3, pp. 159 - 188, 1990.
[17] D. McDermott, "A temporal logic for reasoning about processes and plans," Cognitive Science, vol. 6, pp. 101 - 155, 1982.
[18] T. L. Dean and D. V. McDermott, "Temporal data base management," Artificial Intelligence, vol. 32, pp. 1 - 55, 1987.
[19] T. Dean, R. J. Firby and D. Miller, "Hierarchical planning involving deadlines, travel time, and resources," Computational Intelligence, vol. 4, no. 4, pp. 381 - 398, 1988.
[20] T. Dean and M. Boddy, "An analysis of time-dependent planning," in Proceedings of Seventh National Conference on Artificial Intelligence, Minneapolis, Minnesota, 1988, pp. 49 - 54.
[21] B. Hayes-Roth, "A Multi-Processor Interrupt-Driven Architecture for Adaptive Intelligent Systems," Stanford University - Research Report: June 1987.
[22] B. Hayes-Roth, R. Washington, R. Hewett, M. Hewett and A. Seiver, "Intelligent real-time monitoring and control," in Proceedings of Eleventh International Joint Conference on Artificial Intelligence, 1989.
[23] B. Hayes-Roth, "Architectural foundations for real-time performance in intelligent agents," The Journal of Real-Time Systems, vol. 2, no. 1/2, pp. 99 - 126, 1990.