Advanced Real-Time Embedded System Tools For Intelligent Architectures (ARTESIA)

Michael Harrelson
David Zaleta
Dan R. Ballard
Reticular Systems, Inc.
4715 Viewridge Ave. #200
San Diego, CA 92123

Abstract
Existing knowledge-based systems and development tools were never designed for real-time embedded computing environments since they have historically been used in non-real-time applications. Furthermore, fundamental design concepts for real-time performance have been virtually ignored in the design of languages, shells, and tools. Real-time knowledge-based systems pose unique engineering problems which must be identified in order to resolve these shortcomings. Our research has focused on developing a set of tools to support three real-time AI processing objectives:

1. real-time intelligent control of system resources;
2. predictable real-time knowledge-based system performance; and
3. demonstration of these technologies in an embedded environment running a real-time operating system.

Reticular Systems, Inc. has developed a set of software tools for building real-time intelligent systems. This toolset is called *ARTESIA: Advanced Real-Time Embedded System Tools for Intelligent Architectures*. These tools include a low level real-time executive for intelligent control called TaskMaster™. TaskMaster™ works with the real-time operating system (RTOS) and the knowledge-based programs that run on the RTOS to ensure that the most critical application tasks are accomplished prior to their respective deadlines and that all tasks are completed in a timely manner. TaskMaster™ accomplishes intelligent control by utilizing high-level knowledge about the application program's goals and beliefs as well as the relative priority of the applications. This information is used to generate schedules that meet the application tasks' deadlines. To demonstrate predictable run-times in production systems, we have developed two new expert system software tools: CLIPS-RT and CLIPS-RT Profiler. CLIPS-RT is a version of the CLIPS expert system shell (originally developed by NASA) that has been modified to operate in real-time embedded environments and provide dynamic worse-case run-time information. CLIPS-RT Profiler is a tool that allows the user to determine estimated response times to certain inputs and permits the user to alter the rule base to improve real-time performance.

To adequately test TaskMaster™ and CLIPS-RT, we have integrated TaskMaster™ with a sophisticated situation assessment (SA) expert system into a VME embedded computer running a standard RTOS (VxWorks). SA is a sophisticated rotorcraft decision aide that permits the detection and recognition of external entities and infers high-level attributes about these entities [1-3].

Introduction

Reticular Systems' research has shown that it is necessary to first identify and solve the unique engineering problems associated with *real-time knowledge-based* applications in order to implement real-time knowledge processing systems. This requires recognizing that knowledge-based processing is fundamentally different from other problem-solving paradigms. That is, techniques developed for implementing non-real-time knowledge-based systems may not be applicable to real-time knowledge-

based systems, and techniques developed for real-time systems programming (i.e., real-time kernels, operating systems, etc.) may not be applicable to real-time knowledge-based systems processing. The basis of a real-time knowledge-based system will certainly be based on both knowledge-based and real-time systems. However, the requirements for real-time knowledge-based systems dictate that new design approaches be developed. The rapidly changing, time-dependent nature of real-time events requires the development of fundamentally different programming methods and knowledge management techniques for real-time knowledge-based problem solving.

Real-Time Systems

Traditionally, computer science has defined a real-time system as a system that must not only provide the *correct* answer but provide it within strict time constraints. However, this definition is not adequate for knowledge-based processing systems. AI researchers have no generally acceptable definition of *real-time* knowledge-based processing. Many have assumed that faster execution is an acceptable method for attaining real-time performance. For example, Marsh, concedes that:

> "*A system exhibits real-time behavior if it is predictably fast enough for use by the process being serviced*"[4].

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*" [5].

By nature, knowledge-based systems must be, adaptable to both their external and processing environments. The complexity of these systems prevents *a priori* knowledge of their behavior. Therefore, it cannot be assured that the system will arrive at the one *correct* answer within a certain time constraint. Moreover, with real-time knowledge-based systems, rigorous performance requirements must be met [6]. In these real-time systems, the required response time is a major influencing factor on the overall system design. A critical event cannot be overlooked because the system is processing another action; awareness and responsiveness are essential [7].

A real-time knowledge-based system must be aware of the timing constraints placed on it. The system must be able to make decisions concerning time and integrate time elements into its planning (temporal reasoning). The system must be able to handle elaborate plans in sequence and gracefully degrade when the planning process is time limited. Human experts reason about problems at different levels of abstraction and provide different response times based on the level of their reasoning. Real-time knowledge-based systems must also use this kind of reasoning because it allows the systems to make the best possible decision in the available time.

A more suitable definition for real-time knowledge-based processing is therefore proposed: A real-time system is a system that guarantees an *acceptable* solution within the available time constraints. The acceptability of the solution and the available time constraints are determined by the system designer and the application.

Knowledge-Based Systems

Knowledge-based systems are often called *expert systems* or, in some cases, *artificial intelligence* (AI) systems. Neither of these terms provides an adequate description. While knowledge-based systems do capture the expertise of certain domain experts (e.g., pilots in the case of the Army's Rotorcraft Pilot's Associate program), they consist of much more than just the expertise of a select few individuals. Likewise, the term "artificial intelligence" is too broad to be meaningful. Other technologies, such as fuzzy logic, neural networks, machine learning, etc., must also be encompassed by this term. A knowledge-based system is a system in which human knowledge is captured, represented in some intermediate

form, and then inferences are drawn about this knowledge. Thus, it is important to realize that the primary issues are concerned with how these knowledge representations are structured and how they are reasoned about. There are a number of ways of designing and implementing these knowledge-based systems. These include rule-based systems, predicate logic based systems, object oriented based systems, and frame-based systems [8].

Each of these approaches (and there are a number of others not listed) has its proponents. However, it must be noted that every case is essentially addressing issues of inferencing and representation. Thus, our research has focused on knowledge representations that are efficient for real-time use and inferencing procedures which are amenable to real-time requirements.

A simplified system organization is shown in Figure 1. This figure shows a layered approach that outlines the general design of a real-time AI system.

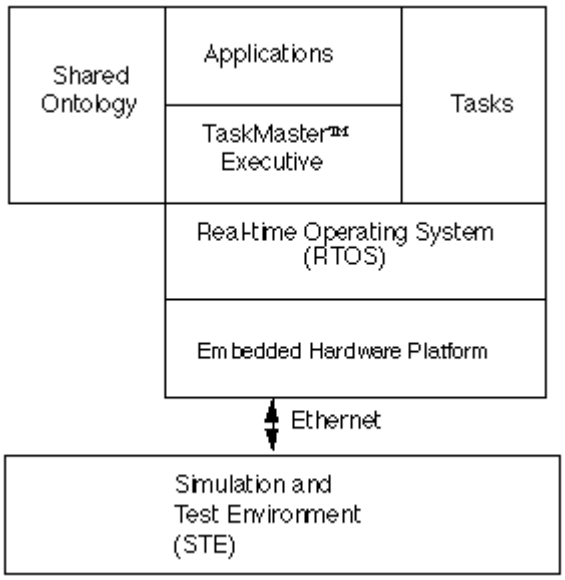

Figure 1 *Overall Real-Time Knowledge-Based System Architecture*

At the lowest software level, there is the real-time operating system (RTOS) which is responsible for handling all of the low-level operating system calls. (The RTOS used in our research was VxWorks.) RTOSs differ from general operating systems (OS) in that they are smaller in size (typically 100-500 kbytes) and have predictable and fast response time (~ msecs) to external events via hardware assisted interrupts.

The TaskMaster™ executive executes on the RTOS. This executive is responsible for *intelligently* controlling the resources of the computer, both in terms of computing time and memory. TaskMaster™ utilizes intelligent control techniques to accomplish this. This means that the executive can dynamically generate a task schedule that will maximize the number of tasks that meet their respective deadlines in a rapidly changing environment. Currently, RTOS scheduling executives are extremely poor in dynamically altering their scheduling algorithms to adjust to complex dynamic environments. To accomplish this, it is necessary to utilize a knowledge base (or "shared ontology") that is shared between TaskMaster™ and the applications. The shared ontology gives TaskMaster™ the ability to *temporally* reason about the various tasks to be scheduled, as well as their relative importance, and then schedule all tasks to meet their deadlines. In addition to using knowledge-based real-time scheduling techniques, logic is incorporated which allows TaskMaster™ to alter its performance/scheduling overhead during system execution.

Conceptually, the embedded system consists of knowledge-based applications that conduct reasoning over the course of a deployment and periodically, or aperiodically, request the spawning of tasks from TaskMaster™. TaskMaster™ uses the shared ontology to reason about when tasks or applications should run and develop a task schedule. This schedule is then executed by TaskMaster™ using services provided by the underlying real-time operating system (RTOS) to meet the required deadlines. A more detailed system description is shown in Figure 2. This figure describes the interaction between the various modules and one of the important global data structures: the *shared ontology*.

The shared ontology provides information about the relative importance of different kinds of application goals, whether the task is periodic or not, past task run-times, and estimated run-times for various tasks given the current environment and knowledge base. TaskMaster™ uses this information to create a schedule, which is then executed at the appropriate time and priority by low-level system calls to the RTOS that spawn the appropriate task or application. The tasks are requested by the applications using intertask communication techniques such as shared memory or message queues. Software interrupts (signals) or real-time intertask communication techniques (event semaphores, message queues, etc.) can trigger new information to transfer between TaskMaster™ and/or a particular application.

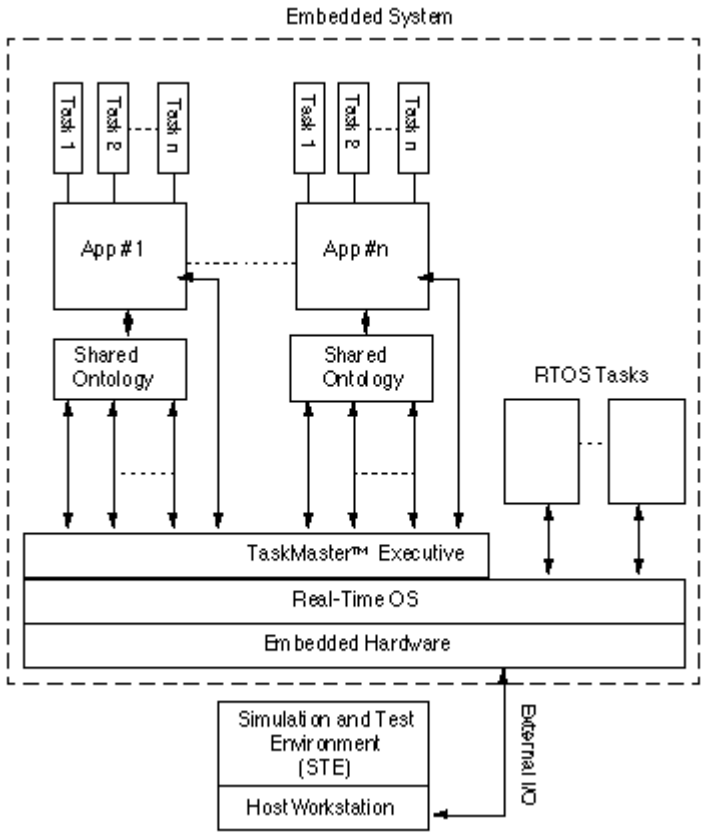

***Figure 2** Overall System Architecture*

Knowledge-based applications require a special dedicated real-time inferencing system for execution in an embedded real-time system. CLIPS-RT provides complete facilities for developing a knowledge-based application. We have developed two versions of CLIPS-RT: one for application development and another for embedded run-time operation. The version of CLIPS-RT used for application development has an X-windows user interface and runs in an interpretive mode on a UNIX workstation, while the embedded version has no user interface and provides compiled rules linked together with the main expert system software. The changes to the standard CLIPS distribution modify the software to provide predictable run-time performance.

TaskMaster™ Real-Time Executive

The TaskMaster™ component is the intelligent controller for the system that allocates resources to the various applications. Applications can vary from traditional procedural programs to complex, artificial intelligence programs, such as a situation assessment application. These applications communicate with TaskMaster™ to obtain the system resources they require. In a real-time environment, there is often a state of overload, meaning that the demand for resources exceeds those that are currently available. Intelligent control of system resources requires that the most important tasks be given the resources they need to accomplish their goals; this is of paramount importance in any real-time system.

Some of the details that TaskMaster™ must consider in order to intelligently allocate system resources include:

- The current goal and state of each application.
- The current tasks being executed and their states.
- The current state of the RTOS.
- External input and output.

TaskMaster™ is modeled as an intelligent software agent. An intelligent agent is usually viewed as a computational process with a single locus of control and intention. Intelligent agents can be divided into several key sections: a *perception subsystem*, a *cognitive subsystem*, and an *executor subsystem*. The perception subsystem is responsible for input into TaskMaster™. Any type of information or request must go through this subsystem. The cognitive subsystem will maintain a knowledge base of the state of the system, process the information from the perception subsystem, adjust to new circumstances, reason temporally, integrate feedback, and handle planning. The executor subsystem is responsible for all external output from TaskMaster™, executing the plans created by the cognitive subsystem, and providing feedback on how the plans are proceeding.

Another key element to TaskMaster™ is the shared ontology. This shared ontology allows TaskMaster™ to access an application's internal information. The shared ontology is a highly-structured knowledge hierarchy that allows knowledge to be shared efficiently and easily between TaskMaster™ and the application.

Architectural Considerations

The key to understanding the problem of real-time performance is to view it as a problem in real-time *control* -- focusing resources *intelligently* instead of using bigger and faster computers. No matter how fast computers become, there will never be a substitute for intelligent use of the resources.

There are three different approaches to combining artificial intelligence with real-time control. The first approach is to embed the artificial intelligence into the real-time system. Unfortunately, this approach creates a temporal bottleneck in the execution of the system. Artificial intelligence techniques are inherently unpredictable and can cause the real-time system to lose its hard deadline guarantees.

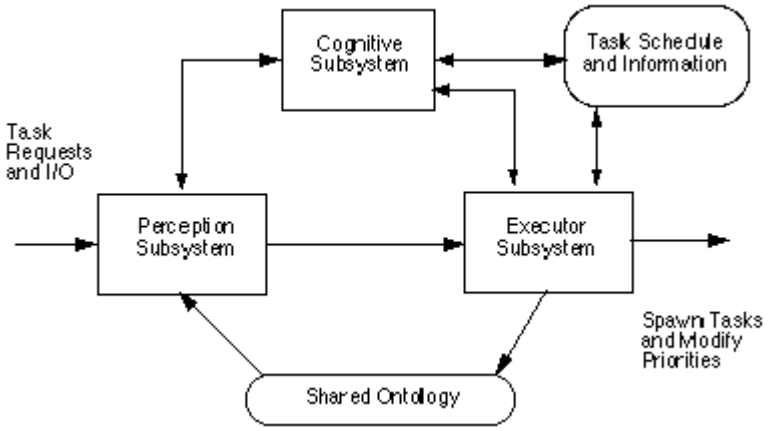

***Figure 3** TaskMaster™ Block Diagram*

The second approach is to embed real-time techniques into the artificial intelligence system, thus, trying to make unpredictable routines more predictable. This approach applies various constraints to their methodologies in order to guarantee the hard deadlines. The problem with this approach is that it puts an AI system into the control cycle of the system which can introduce serious delays and timing problems [9].

The third technique involves *separating* the artificial intelligence and real-time components. This strategy involves integrating the two techniques into one system, but not placing the artificial intelligence into the real-time control cycle of the system. In this way, real-time reactivity can be achieved without slowing down the AI system. The difficult part of the problem is effectively using the intelligence of the AI system.

The strategy with TaskMaster™ is to put the artificial intelligence into a reflective component of the system. This reflective subsystem runs outside of the main control flow of the system. This allows for

higher level reasoning while, at the same time, not breaking the real-time guarantees associated with the system.

TaskMaster™ Architecture

Figure 3 displays the basic set of subsystems that make up the TaskMaster™ system. All of the subsystems are individual components that run as individual tasks independent of one another. This independence allows the subsystems to operate simultaneously, thereby allowing more efficient utilization of the system's resources. The major components of TaskMaster™ include:

- *The Shared Ontology* -- This subsystem gives TaskMaster™ the ability to understand the application's goals and the associated techniques required to accomplish these goals.
- *The Perception Subsystem* -- This subsystem handles the processing of incoming messages from and about the external and internal environments and includes an overload reactive component.
- *The Cognitive Planning Subsystem* -- This subsystem uses the information contained in the shared ontology to develop a scheduling plan.
- *The Executor Subsystem* -- This subsystem implements the planned schedule and communicates with the external and internal environments. It contains a list of currently spawned tasks that the executor subsystem manages. These include the suspended, stopped, and currently running tasks (including itself).

CLIPS-RT and CLIPS-RT Profiler

CLIPS (C Language Integrated Production System) is an expert system tool developed by the Software Technology Branch, NASA/Lyndon B. Johnson Space Center. CLIPS is a knowledge-based systems shell that implements a production system. CLIPS can be called from a procedural language, perform its function, and then return control back to the calling program. Likewise, procedural code can be defined as external functions and called from CLIPS. When external code completes its task, control returns to CLIPS [10].

A production system program consists of an unordered collection of IF-THEN statements called productions. The data operated on by productions is held in a global database called working memory. By convention, the IF part of a production is called its LHS (left-hand side), and its THEN part is called its RHS (right-hand side). The interpreter executes a production system by performing the following operations:

- *Match* -- Evaluate the LHS of the productions to determine which ones are satisfied given the current contents of working memory.
- *Conflict Resolution (Select)* -- Select one production out of possibly several with a satisfied LHS.
- *Act* -- Perform the actions in the RHS of the selected production.
- *Repeat*

An object together with its attribute value pairs is called a working memory element. The LHS of a production consists of a sequence of patterns; that is, a sequence of partial descriptions of working memory elements. When a pattern P describes an element E, P is said to match E [11].

The pattern matching operation in production systems is an unpredictable element that can significantly affect real-time performance. This is true even though CLIPS (and most other modern production systems) utilize some form of the Rete algorithm to minimize the matching time required. There are two factors that make a production system unpredictable. First, a single assertion can produce an avalanche of data cascading down the Rete pattern matching network. Second, the firing of a single rule can result in the firing of additional rules which can assert more data to be pattern-matched. In fact, it is possible that a single assertion or rule activation could continue indefinitely. To handle the first case, we have

implemented a dynamic method of determining the run-time to assert any given fact. In the CLIPS-RT Profiler description, we present our approach to handling the second case.

CLIPS-RT Modifications

To make CLIPS more usable by real-time systems, the following features were incorporated into CLIPS-RT.

- Removal of unnecessary user interface, removal of run-time parsing overhead, and addition of a continuous run capability.
- Incorporation of real-time operating system calls to input real-time data from sensors and communicate information to and from TaskMaster™.
- Addition of temporal reasoning functions based on interval based temporal logic [12].
- Division of CLIPS-RT into modular tasks allowing communication, I/O, and primary CLIPS functions to operate as separate tasks but allowing passing of data via RTOS shared memory.
- Incorporation of capability to generate worst-case assertion/retraction run-time information (see below).

The capability to generate worst case run-time estimates represents the most extensive and potentially useful of all the additions/alterations made to CLIPS-RT. It is necessary to develop a reliable run-time analysis of assertions in CLIPS-RT. This requires a detailed knowledge of the inner workings of CLIPS-RT's pattern matching code and internal data structures that implement the Rete algorithm. In order to develop dynamic run-time information about the matching process of CLIPS, we developed a number of tools.

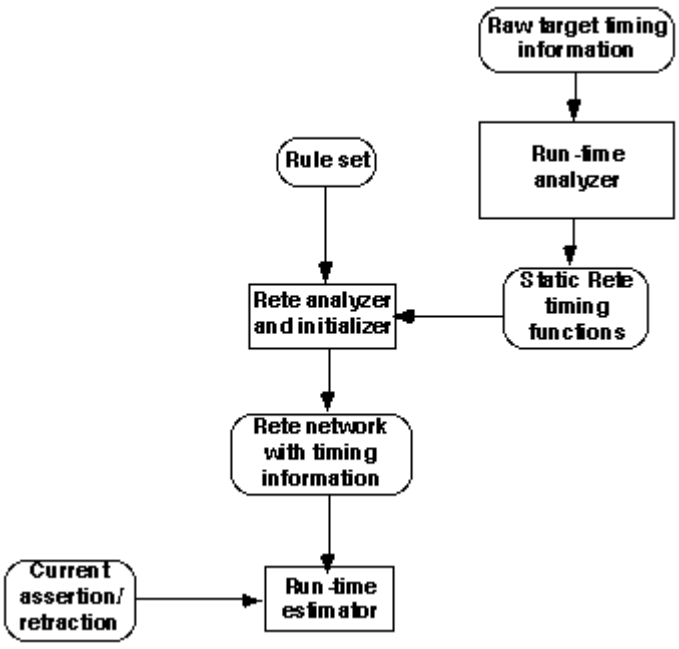

Figure 4 *Design of run-time estimator*

Figure 4 shows the design for the run-time estimator and the associated software modules that were developed. The basic principal of this design is to create as much information as possible at initialization time so that minimum computation is required to compute the worst-case run-time for the Rete network at run-time.

In order to estimate the run-time, either source code or assembly code level timing analysis is required. For simplicity, we analyzed the source code by decomposing each C statement into its constituent parts and then collecting timing information for each of those operations on the target CPU (in our case a 68060 VME board). This information consists of the time required to do simple C operations of such statements as `if`, `for`, `while`, `assign`, `dereference`, function calls, etc. This data is then put into a data structure that could be directly used to calculate the worst-case run-time of any C statement.

The next step was to create initialization C files that calculate the worst-case time that CLIPS-RT C functions (specifically those functions responsible for assertion/retraction in CLIPS-RT) take to run. This requires a decomposition of all functions that implement the rete network assertion/retraction process. During this project, a tool called the Run-time analysis module was created to aid the user in generating these files. These initialization driver functions then initialize the timing information in each join node of the rete network. Note that assertions into a NOT join node can cause retraction of some partial matches further down the network. So both the assertion run-time costs and retraction run-time costs are required.

There are fixed and variable times associated with asserting or retracting a fact (and all partial matches containing that fact) from the rete network. The fixed costs consist of a constant overhead for driving a token through a join node, whereas the variable costs are dependent upon the number of partial matches in the alpha/beta memories. Currently, worst case pattern matching results (all new tokens match with all partial matches in either the alpha or beta memory, depending on the direction of driving) are assumed.

Once the estimated join node timings are generated and placed into the appropriate data structures, it is then possible to create the run-time estimator. This function can be called at run-time to calculate an estimate of the worst-case time it takes to assert a given fact into the rete network based on the current state of the network.

Comparisons between measurement of actual run-times and estimated run-times generated by this method show that in all cases the estimated run-times exceed the actual run-time. Because of some of the assumptions that are necessary, the estimated run-times can be significantly higher than actual run-times. This is mainly because the pattern nodes have a significant variance in the code that is actually called upon at a given time (generally due to the presence of multifields).

CLIPS-RT Profiler

To solve artificial intelligence problems in real time, it is necessary to make solution times predictable. In other words, tasks requested by CLIPS-RT must be associated with a realistic estimate of the run-time that the specific task will require. Two methods of accomplishing this were identified: (1) placing limitations on the number of join matches (or other Haley constraints) and conducting a Haley analysis; and/or (2) maintaining a consistent run-time estimate (perhaps updated by prior run-times) based on the current working memory and rule base.

The Haley constraints and the associated analysis algorithms were first proposed by Haley [13]. We have implemented a version of the Haley algorithm into CLIPS-RT Profiler. Haley presents several methods for bounding the cost of matching a rule, including join node match limitations, pattern instantiation restrictions, relation instance restrictions, and cardinality restrictions. This section describes the Haley constraints in the CLIPS-RT Profiler. Note that CLIPS-RT is designed to run on an embedded computer, while CLIPS-RT Profiler is designed to create an environment on a standard workstation for development of real-time CLIPS-RT code.

One of the main attributes of CLIPS-RT Profiler is to allow users to evaluate rules written in CLIPS-RT to determine their suitability for the intended application. This happens during development and before embedding the application into the real-time computing environment. In addition, the profiler allows the user to graphically examine the Rete network to see if other optimizations can be performed on the network or the rules to speed up and support more predictable real-time operation of the knowledge base. Therefore, the two main features that CLIPS-RT Profiler must provide:

1. Haley worst-case run-time analysis.
2. Graphical interaction with rete network and rule activation network.

The CLIPS-RT Profiler is the vehicle that provides the functionality needed to obtain the worst case run-time given that a rule base has certain constraints placed on the number of join matches. In order to help the user make good decisions about where to place these limitations, a graphical tool for viewing the Rete network as well as a graph of the rule activation network is included. This functionality has all been directly included into the CLIPS-RT Profiler and can provide the user with an abundance of information concerning the run-time of the rule base.

Before we can evaluate the run-time of the rule-base, it is necessary to build some representation of the activation connections between all rules in the rule set. It is also necessary to check for cycles that might cause infinite rule activation. There are two distinct phases of the rule activation graph building process: (1) create and link all rule structures and (2) cycle checking. The first phase is responsible for building all information required to represent the activation connectivity between rules.

To build this connectivity, we need to look at each assertion and retraction on the RHS of each rule. Then we search all other rules to see if there are any matches between the template being asserted and a template on the LHS. Matches occur if the template type being asserted/retracted is the same as one of the LHS conditional elements in another rule (possibly the same rule). If a match exists, then a link from one to the other is created. As each new rule is visited the function is called recursively to traverse the interconnected rules.

The second phase is to traverse this graph and determine if any cycles exist. This is accomplished with a standard depth-first traversal at each step, marking visited nodes and assigning each node a number that corresponds to when the node was visited in the depth-first search. A cycle is detected if one of the links from a rule graph node points to a node that has already been visited (it has a smaller number associated with it) and is an ancestor of the current node. If a cycle exists, then there is no predictable run-time estimate that can be derived, since an infinite number of rule activations is possible. If a loop does not exist, then a worst-case run-time can be calculated for the given assertion. Note that once a loop is detected all patterns in the LHS of all rules in the loop have unpredictable run-times since the assertion or retraction of any of these could potentially lead to an infinite loop condition.

Note that the purpose behind the limitations posed by the Haley constraints is to restrict the number of potential matches that can occur at each join node. Therefore, the constraint that is the smallest at each join node is the limit placed on the number of matches at that node. The current version of the CLIPS-RT Profiler does not directly support the cardinality restrictions. This is because the current version considers only the pattern name for Haley analysis matching purposes. Therefore, there is no way to distinguish between different cardinality restrictions that might exist for different subsets of constants in a pattern's fields or slots. Future versions will provide support for this constraint type.

Once the above limits are determined for all join nodes in a knowledge base, the Haley algorithm allows the calculation of the bounded cost of asserting any given relation by simulating the execution of the rules with the given join match limits. A method (based on Haley's algorithm) was developed to generate realistic run-time estimates based on a specific input event. Note that this analysis is done completely during the final design phase and is not generally active at run-time on the embedded system. However, although it is not currently implemented, the design could readily be extended to run-time execution even though the overhead involved in this could be potentially prohibitive.

Results

In order to test ARTESIA, we created a test environment consisting of a VME 68060 single board computer, a Sun Sparc10 host workstation, and an IBM RS/6000 workstation (all of which are connected via an ethernet). TaskMaster™ and a CLIPS-RT version of our rotorcraft situation assessment (SA) expert system were integrated onto the VME computer from the host workstation. In order to run SA, a simulation and test environment (STE) program was executed on the IBM workstation. The STE modeled

the dynamic environment of the rotorcraft and communicated sensor information to SA via a socket connection. We successfully demonstrated the integration of this system and verified that the results were compatible with previous results for a non-real-time version of the system.

During this project we have created a basic set of real-time tools for aiding the development of real-time knowledge-based systems that must run in a predictable manner. A real-time executive, TaskMaster™, was developed that controls the access to computational resources for competing processes. Using a high-level shared ontology knowledge base, TaskMaster™ is able to allocate resources based on the current state of the environment as well as the current priority of the competing processes. These processes can be an application program (e.g., situation assessment, mission planning, pilot-vehicle interface, etc.) coded in a production system such as CLIPS-RT. We have modified CLIPS to create a real-time embedded version (CLIPS-RT) that is able to predict the worst case performance requirements. In addition, we created a real-time development environment, CLIPS-RT Profiler, capable of analyzing rule bases to determine total worse case run-times give the activation of a certain rule or assertion of a given fact. This research has shown that it is possible to build real-time knowledge-based systems, predict their worst-case run time performance, and orchestrate and control multiple applications in an embedded computer system. We plan to extend the capabilities of ARTESIA in future versions.

References

[1] D. R. Ballard and L. Owsley, "Situation assessment in rotorcraft associate systems," in *Proceedings of Associate Technology Conference: Opportunities and Challenges*, Fairfax, VA, 1991, pp. 111 - 121.
[2] D. R. Ballard and D. J. Nielsen, "A real-time knowledge processing executive for Army rotorcraft applications," in *Proceedings of 11th Digital Avionics Systems Conference*, Seattle, WA, 1992, pp. 199-204.
[3] T. Bryson and D. Ballard, "PalymSys™ - An extended version of CLIPS for constructing and reasoning with blackboards," in *Proceedings of Third Annual CLIPS Conference*, Houston, TX, 1994, pp. 377 - 387.
[4] 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.
[5] D. A. O'Reilly and A. Cromarty, "'Fast' is not real-time," in *Applications of Artificial Intelligence II*, vol. (Ed.) Internation Society of Optical Engineering, 1985, pp. 249 - 257.
[6] L. L. Odett and W. B. Dress, "Engineering intelligence into real-time applications," *Expert Systems*, vol. 4, no. 1, pp. November, 1987.
[7] M. Wright, M. Green and P. Cross, "An expert system for real-time control," *IEEE Software*, vol. no. 3, pp. 16 - 24, 1986.
[8] D. A. DeSalvo, "Introducing AI into the Systems Development Model," in *Structuring Expert Systems*, vol. J. Liebowitz and D. A. DeSalvo, (Ed.) Yourdon Press, 1989, pp. 333 - 356.
[9] D. Musliner, E. Durfee and K. Shin, "CIRCA: A cooperative intelligent real-time control architecture," The University of Michigan, Department of Electrical Engineering and Computer Science, 1993.
[10] NASA, "CLIPS Reference Manual: Volume I - Basic Programming Guide for CLIPS Version 5.1," NASA Software Technology Branch, Lyndon B. Johnson Space Center - JSC-25012, September 10, 1991.
[11] C. L. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. pp. 17 - 37, 1982.
[12] J. F. Allen, "Towards a general theory of action and time," *Artificial Intelligence*, vol. 23, no. pp. 123 - 154, 1984.
[13] 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, pp. 277 – 282.[1]

[1] This research was supported by the U.S. Army Aviation Applied Technology Directorate (AATD), Fort Eustis, VA, under contracts DAAJ07-92-C-0020 and DAAJ02-95-C-0008