In the last several years, a great deal of research has been expended on the development of real-time embedded Artificial Intelligence (AI) systems. One of the major problems associated with these systems is achieving real-time performance. An early approach to this problem was to use a centralized control structure, such as the blackboard architecture, to manage the memory and processing utilization for the intelligent system. Unfortunately, the centralized scheduler placed an intensive computational burden on the real-time system. This resulted in research directed towards decentralized control architectures, such as the distributed blackboard or communicating expert objects. One such communicating expert object paradigm is the Activation Framework (AF) architecture. This architecture was utilized by the Adaptive Tactical Navigation (ATN) program, a laboratory demonstration of an intelligent navigator, which achieved real-time performance. The AF architecture provides many benefits over centralized control architectures including efficient control, modular knowledge base partitioning, and the ability to easily parallelize the system to operate on multiple processors.

This paper will discuss the AF architecture and the Engineering Graphical Analysis Tool (EGAT), the utility of EGAT, and how EGAT fits into the embedded real-time development environment. EGAT provides the user with the capability to analyze and modify the dynamic control characteristics of a running AF system. This tool gives the user methods for examining the interaction between the AF groups, the interaction within the AF group, and detailed information about the AF parameters and communications through a graphical interface. EGAT promises to greatly aid in the development and refinement of real-time AF systems by alleviating the current process of manually analyzing the inter-AF communication.

THE AF ARCHITECTURE

The AF architecture is a communicating expert object system which is ideal for real-time AI systems because of its low overhead for memory management and scheduling functions. It provides a hierarchical structure in which the AF is at the highest level. At the next lower level, are groups called subgraphs. Each subgraph contains several objects called Activation Frame Objects (AFOs). An AFO emulates the behavior of a human expert. It has knowledge about a specific domain; it receives messages from other experts (AFOS) via message input queues and acts upon these messages. When an AFO has acted upon the messages in its input queue, it then sends information to other AFOs in the system. A group of AFOs working on a specific problem is grouped into a subgraph. The subgraph is very useful for focusing the resources of different AFOs on a common goal. A collection of subgraphs, usually on one processor, is called an AF. The AF provides a methodology for the subgraphs to work with each other to solve a complete problem. The AF can be implemented on one processor, or it's subgraphs can be split and inserted into different AFS for multi-processor operation.

The AF architecture uses a decentralized control scheme which is implemented as follows. Each AFO has an importance level which has been initialized to some value. In addition, every AFO message also has a message importance associated with it. This message importance is used to boost the importance level of a receiving AFO. In this manner, as the AFOs receive messages, their overall importance level increases and the AFO with the highest importance level will get the processor to execute next. The AF architecture

U.S. Government work not protected by U.S. copyright 1130
provides flexibility in determining the order of execution and the level of importance each message possesses. This is achieved because the user specifies the initial importance levels of the AFOs which determines the order that the AFOs will execute. For example, if the AF architecture is implementing a rule based system, then the first rule in the system would have the highest importance, the second rule the second highest importance, etc. This way the AF system would run like the rule based system would, with the first rule executing first, then the second rule, sequentially through the rulebase. The user also has the flexibility to specify the importance level of each message determining how much influence each message has on the system. Each AFO contains four functions: the initialization, priming, importance, and transfer functions.

The initialization function is used to create the AFO and its associated input ports. A new entry on an input port table is created for each new input port. In the AF architecture, each AFO can have several input ports associated with it. This provides flexibility for determining when an AFO has the information it needs to make a decision (called "being primed"). Additionally, the initialization function also assigns the global importance to the AFO. This importance level is used to determine the AFO's initial importance level (the importance level without input message importances added).

The priming function is used to determine what messages are needed on the AFO input queues before an AFO can run. An AFO runs when the AFO gets the processor, reads its messages, performs its designed function, and sends out necessary messages. When all necessary information is acquired, it is then primed and waits to become the most important AFO in the system (i.e. the one with highest importance level). The AF architecture uses the priming function to set up "and/or" relationships between the message queues to determine when the AFO is primed.

The importance function determines how the overall importance level of the AFO will be determined. In the AF architecture, only one AFO can run on one processor at a time. Therefore in order to properly focus the system resources, the AFO with the highest overall importance level will get the processor next. The default importance function for each AFO sums up the individual message importances to determine the AFO's overall importance level. However, the user can change the importance function to suit the application.

Finally, the transfer function is responsible for reading in the messages from the input queues, reasoning about the messages, and sending response messages to the AFOs. The AFOs expertise is contained in the transfer function. Although the AF architecture provides a great deal of flexibility in determining the control of the AF system, this flexibility causes difficulties in setting up the system. With the user defined initial AFO levels and each message importance level having such a strong impact in controlling the AF architecture, the determination of each of these importance levels becomes a very tedious task. Not only are the initial levels hard to set but it is also difficult to determine if the system is operating correctly when it is running. Due to the difficulties involved with correctly setting the AF control parameters, a tool was needed to help alleviate this problem. As a result, the Reference Systems Branch developed the Engineering Graphical Analysis Tool (EGAT) through an in-house effort.

EGAT system

The EGAT system is a hierarchical system which allows the user to visualize the operation of a running AF system. This hierarchical system has three levels of graphic displays (See Figure 1). The highest level of the EGAT system shows the subgraphs that make up the AF. This level is called the subgraph level. Each subgraph is made up of a collection of AFOs. At the subgraph level, the message traffic from one subgraph to another is shown graphically by different colored lines between the subgraphs. The highest level of message traffic is represented by a red line and the lowest level is represented by a violet line, following the visible spectrum. All together, there are seven different colors that follow the color spectrum. They are red, orange, yellow, dark green, light green, blue, and violet.
The next level in the EGAT hierarchical structure is the AFO level which represents the AFOs within one subgraph. This display is reached by selecting one of the subgraphs. At the AFO level, each AFO is represented by a graphical circle. These circles are separated into two parts: an outside ring and an inner circle. The inner circle is used to determine the state of the AFO. There are three states for an AFO: a green internal circle represents insufficient input information; a yellow internal circle represents the AFO has sufficient information to execute, however the importance level is too low to execute; and a red internal circle means the AFO is executing. The outer AFO ring represents the importance level of the AFO. This importance level is represented by the same colors as used to show the amount of message traffic between the subgraphs. The actual importance level is represented as a integer value which appears inside the circle. In addition to the AFOs, there are also multiple graphic lines connected to the AFO circles which represent the actual AFO input ports. When an AFO message is sent within the same subgraph, a black line will flash between the center of the sending AFO to the tip of the appropriate AFO input port line of the receiving AFO. In this way, the EGAT user can determine exactly where the message was sent.

The lowest graphic display level, the AFO information level, is reached by clicking on one of the AFOs. From this level detailed information about one AFO is displayed. This information includes the AFO name, the AFO’s subgraph name, the AFO number, the AFO’s importance level, the clock time in milliseconds from when the AF began running, the AFO’s priming state, and the actual messages on the AFO input queues. The AFO and subgraph names are the names assigned to the AFO and subgraph respectively. The AFO number is an internal EGAT number that is used by the AF architecture to uniquely identify the AFO. The clock and primed status are self.
The real power of the EGAT system is its ability to dynamically change the importance level of a running AFO or to inject or delete messages from their input queues. By double clicking on the AFO importance box, the EGAT user can dynamically change the importance level. This allows the user to change the importance level of an AFO and then see what effects this action has on the rest of the system. Likewise the AFO messages can be examined or changed in the same fashion by double clicking on the appropriate message input queue. For larger messages, two arrow boxes "<--" and "-->" are provided to allow the user to scroll the message across the screen. In addition, the "prev" and "next" boxes are provided to scroll between different messages in the same input queue.

At the four corners of every level are the EGAT action boxes, halt, step, quit and up. The halt box allows the user to temporarily suspend the operation of the running AF system in order examine the static state of the AF system. While the system is halted, the user can use the step action box to single step through the system. The up box provides a mechanism to go to higher display levels. The up box does not serve any purpose at the subgraph level. The quit box provides a mechanism to stop and quit the AFO system at any point.

UTILITY OF EGAT

EGAT, with its strong graphical interface, allows the user to see the dynamic inter-workings of the AFOs, and assists the designer with many aspects of the development of the AF system. Before the EGAT system was developed, the user was forced to manually trace the inter-AFO messages to determine if the system was functioning as desired. This was a very time consuming and tedious task, because the designer, would have to analyze the activity of the system with only the AFO message history. Once a problem was discovered the designer would adjust the control parameters, rerun the system and again analyze the inter-AFO messages for correct operation. Now, by using the EGAT system, significant time savings are achieved because the messages and sequence of message traffic can be seen in real-time on the screen. If there is a problem the designer can analyze the control parameters as the system is running. This is extremely important to real-time AI designers due to the fact that the control parameters depend heavily on specific domain knowledge. Non real-time AI designers also have difficulties correctly tuning their system. EGAT aids in this process by allowing step by step execution of the system and the examination of the importance levels of the AFOs and messages. This fine-grained execution can help in determining how the importance levels should be changed to provide a correct and more efficient system. Not only can these importance levels be examined, but through EGAT the levels can be changed while the system is running. After the change is made the system can be continued, thus determining the effects of the changes.

EGAT'S ROLE IN THE EMBEDDED REAL-TIME DEVELOPMENT ENVIRONMENT

As described in this paper, EGAT is a stand alone tool which can be used to dynamically monitor and modify an executing AF application. In addition, EGAT was developed to be part of an embedded real-time AI development environment. This real-time development environment is being jointly developed by the Air Force and NASA, and it includes the following programs: the Knowledge Representation into Ada Methodologies program, the Knowledge Representation into Ada for Parallel Processing program, and the Shell for Knowledge Representations into Ada Methodologies program. The culmination of this effort will result in an efficient user friendly environment that will automatically verify and validate the knowledge base, convert the knowledge base into Ada, and efficiently host the application onto parallel processors in order to provide the necessary throughput in the system to achieve real-time performance. At this time, in order to better understand this embedded environment, the individual components will be discussed.

The Knowledge Representations into Ada Methodologies (KRAM) program is responsible for developing knowledge base verification and validation techniques, as well as, automatic methods to convert knowledge representations into Ada code. KRAM takes knowledge representations such as forward and backward chained rules, ART and CLIPS representations, and unification type knowledge into an intermediate form called Evidence Flow Graphs. The Evidence Flow Graphs are a graphical representation
which is utilized to perform Monte Carlo testing on the AF architecture to insure stability and graceful degradation of the system. After the rules are satisfactorily tested, they are automatically converted into Ada AFOs. These AFOs can then be inserted into the AF architecture to provide an efficient run-time environment. Once in the AF architecture, the EGAT tool can then be used to provide for dynamic testing and tuning of the AF system. In this way, the KRAM verification and validation checking will be used in conjunction with the EGAT tool to test for errors in the knowledge base and insure the correct operation of the run-time system.

Once the system has been converted into an Ada based AF system, the Knowledge Representations into Parallel Processing program focuses on techniques to efficiently host the application onto parallel processors. Under this effort techniques were developed to automatically and efficiently place AFOs onto different processors to enhance real-time performance. Several different load balancing routines were developed to enhance the parallelism in the system and to minimize the dependency between the different AFOs. In addition the effects of adding fault tolerance to the system were also measured to insure stable operation and recovery of the system without sacrificing real-time performance. The EGAT tool is very useful for distributing AFOs onto different processors because the user can easily examine the amount of inter subgraph communication between the AFOs and take steps to minimize it.

The integration of all of these separate tools and techniques into one uniformed package will be accomplished by the Shell for Knowledge Representations into Ada Methodologies (SKRAM) program (See Figure 2). This effort will combine the work done in these previous efforts into a complete embedded real-time environment. This overall tool will also provide graphical interfaces to facilitate the ease in use of the tools. When the overall effort is completed, an integrated embedded real-time environment will be
developed that will allow real-time AI designers to easily develop and maintain their applications. The system will validate, convert into Ada, and parallelize the knowledge representations into an efficient run-time system.