WL-TR-91-1063

2

EXPERT SYSTEM MANAGEMENT SYSTEM

Dr. Arthur Gerstenfeld
Kiet D. Hoang
UFA, Inc.
335 Boylston Street
Newton, MA 02159

30 AUGUST 1991

FINAL REPORT FOR PERIOD JANUARY 1987 - SEPTEMBER 1990

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

NOTICE

Barbara L. Eldridge

Barbara L. Eldridge
Electronics Engineer
Systems Group

[illegible]

DAVID A. ZANN, Acting Chief
System Integration Branch

FOR THE COMMANDER

[illegible]

CHARLES H. KRUEGER, Chief
System Avionics Division
Avionics Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for Public Release; Distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
WL-TR-91-1063

6a. NAME OF PERFORMING ORGANIZATION
UFA, Inc.

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Avionics Directorate
WL, AFSC

6c. ADDRESS *(City, State, and ZIP Code)*
335 Boylston Street
Newton, MA 02159

7b. ADDRESS *(City, State, and ZIP Code)*
WL/AAAS-2
Wright-Patterson AFB, OH 45433-6543

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
Contract No. F33615-C-88-1720

8c. ADDRESS *(City, State, and ZIP Code)*

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
65502F	3005	10	88

11. TITLE *(Include Security Classification)*
Expert System Management System

12. PERSONAL AUTHOR(S)
Gerstenfeld, Arthur, Dr. and Hoang, Kiet D.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Final	FROM 01/87 TO 09/90	91 AUGUST 30	43

16. SUPPLEMENTARY NOTATION
None

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
01	02	
12	07	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Expert Systems, Pilot Vehicle Interface, Artificial Intelligence, Multiprocessor, Message passing

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The Expert System Management System (ESMS) Small Business Innovative Research Contract developed a distributed fault-tolerant expert system shell for multiple expert systems in a multiprocessor environment. The ESMS contained four domain specific expert systems called Manager Expert System, Route Planner Expert System, Weapon Expert System, and Situation Awareness and Display Expert System. The ESMS expert system shell was written in LISP and runs on four Explorer II computers. Each of the expert systems were responsible for their ow area of expertise, but each communicated and cooperated with the other experts along with a real-time man-in-the-loop flight simulator. The four expert systems developed were host independent, relocatable, and interruptable when a higher priority task arrived. The ESMS wa able to detect network and processor failures, along with being able to reconfigure the syste according to the available resources. The system was both event and data driven, and shared global variables. The system was interfaced with a MicroVAX II computer to the MIL-STD-1553B data bus in order to receive waypoints and aircraft data from a real-time man-in-the-loop simulation

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)*	22c. OFFICE SYMBOL
Barbara L. Eldridge	513-255-4827	WL/AAAS

CONTENTS

CONTENTS (CONT'D)

LIST OF FIGURES

ACKNOWLEDGMENTS

We would like to take this opportunity to thank our technical manager, Ms. Barbara Eldridge, and Captain Jesse Fanning for their enthusiastic support. Their input and cooperation during the integration phase of this project was particularly appreciated.

SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
ESMS	Expert System Management System
FIFO	First-In First-Out
GENASIS	Generalized Avionics Simulation/Integration System
ID	Identification
I/O	Input/Output
OS	Operating System
SADE	Situation Awareness and Display Expert
SBIR	Small Business Innovative Research
TCP/IP	Transmission Communication Protocol/Internet Portocol
TI	Texas Instruments
UFA	United Facility Associates

1.0 SUMMARY

This document is the Final Report for the Phase II Small Business Innovative Research (SBIR) project, Expert System Management System (ESMS). It was a 2-year project, which began in January 1987 and ended in April 1990.

This research was initiated as an attempt to incorporate expert system technology[1] in an advanced fighter cockpit to reduce pilot workload and increase mission effectiveness. The increasing complexity of modern avionic systems and mission requirements has imposed tremendous demands on the pilot to make decisions quickly and appropriately. With the help of an expert system such as the ESMS, a pilot will be able to concentrate on the more important tasks while the expert system assumes the routine problem solving tasks.

The focus of this project was to develop a distributed fault-tolerant inference engine enabling multiple expert systems to cooperate and run on one or more processors. In addition, a set of four domain-specific expert systems were developed, the Manager Expert System, Route Planner Expert System, Weapon Expert System, and Situation Awareness and Display Expert (SADE) System. Each of these expert systems while in charge of its own area of expertise communicated and cooperated with the other expert systems, and interfaced with the real-time flight simulator, the Generalized Avionics Simulation/Integration System (GENASIS). The ESMS inference engine was developed in Lisp and ran on four networked Texas Instruments (TI) Explorer machines.

The final ESMS was demonstrated to show near real-time distributed fault-tolerant problem-solving capabilities. It also showed initial pilot aiding in route planning and weapon control. Therefore, the ESMS did accomplish the objective of this research, which was to show real-time distributed fault-tolerant expert system capabilities in an avionic environment. Some valuable

[1] Expert system is a branch of Computer Science which utilizes human expertise and rule-of-thumb to enable a computer to aid in making a decision.

lessons were learned from this SBIR, particularly, that distributed expert system execution is difficult to trace and the use of processes is very important to any real-time expert system application.

2.0 INTRODUCTION

Expert System Management System is a programming tool developed for creating multiple expert systems in a multiprocessor environment. It provides fault-tolerance and dynamic resource allocation capabilities, as well as forward and backward chaining of most traditional Artificial Intelligence (AI) programming languages. It also supports interfaces to MicroVAXs via Transmission Communication Protocol/Internet Protocol (TCP/IP) for real-time data access, such as simulated flight data. The ESMS runs on Symbolics or Texas Instruments Explorer Lisp machines operating under Common Lisp. In the network of Lisp machines, the communication protocol is Chaosnet.

The advantage of ESMS is that it provides an easy and flexible mechanism for multiple expert systems to communicate and share information with each other in a distributed environment. The cooperating expert systems may all reside on the same host or on multiple geographically distributed hosts. In addition, the ESMS also provides a mechanism for near real-time problem solving, such as context-sensitive knowledge base partitioning and time-sensitive functions. As opposed to most other AI tools which were designed for interactive and non-time critical uses, ESMS was conceived and designed specifically for distributed real-time fault-tolerant problem solving. It runs independently with input from sensors and/or interactively with input from a user. The distributed environment under which the ESMS was developed is shown in Figure 1.

Some of the issues addressed in developing a distributed cooperating expert system include:

a. Host Independence. In a distributed environment, the execution of an expert system should be independent of the host on which it resides. In other words, one or more expert systems may run on one or more hosts, and the identity of the host on which the expert resides is unknown until run time.

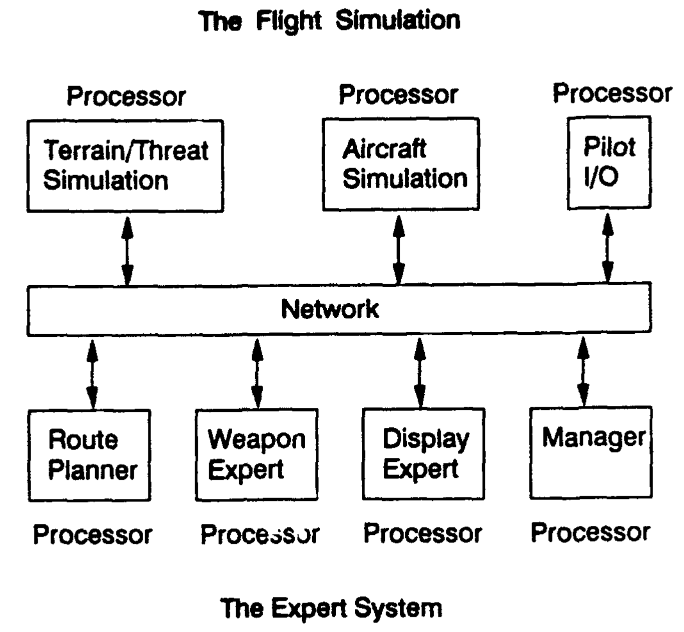


FIGURE 1. THE DISTRIBUTED ENVIRONMENT OF ESMS

b. Control of Cooperating Expert Systems. Since each expert system is running concurrently on its own processor, the system must be able to handle asynchronous events. In the ESMS, communications between different expert systems is handled through message passing. Messages can be sent and received asynchronously; therefore, an expert will never wait for a reply.

c. Focus of Problem Solving. Since events in the real world occur unpredictably, a running expert system must be able to focus on the most critical task/event. In ESMS, the focus of problem solving is based on event priority and interrupts. The execution of an expert system can be interrupted based upon the states of the world or the criticality of an occurring event.

d. Fault Tolerance. Any of the distributed expert systems may fail in a number of ways, such as processor malfunction or network disconnection. Therefore, it is very important that the system possess the ability to dynamically reconfigure and to transfer necessary information among various subsystems. Since the execution of an expert system in ESMS is independent of the host on which it resides, the level of fault tolerance is very high. Processor independence helps make fault tolerance possible, although it is only part of the approach.

e. Problem Solving in Real Time. Due to the nature of the flight domain, it is critical that an expert system communicate with its neighbors in real time and arrive at a solution within given time constraints. In ESMS, a set of timer and default functions are provided to handle time-sensitive problem solving.

f. Problem Solving in Limited Resource. The expert system must be able to make decisions within its own available resources, such as computer time, memory, and communication network load. Each system must be able to intelligently allocate tasks in respect to resources. The ESMS is currently capable of dynamic resource allocation in respect to computer resources. It is not capable of dynamic resource allocation with respect to the problem requirements of the expert system.

3.0 BASIC ESMS CONCEPT

Each expert system in ESMS is self-contained. Each knowledge base in ESMS (except in the Manager Expert System) consists of domain-specific knowledge to handle both internal and external events. Since each expert system is responsible for its own area of expertise, there is no master or slave relationship between any of the four systems. In the case of the Manager Expert System, its primary responsibility is to monitor and fault-recover the other three expert systems. The whole distributed system is very loosely coupled, so that the addition and deletion of any expert system and/or processor will not affect the integrity of the rest of the system.

Instead of production rules found in most conventional expert systems, the knowledge base of an expert system in ESMS consists of event-handlers, daemons, and functions. As the name implies, an event-handler is event-driven. It is a portion of procedural knowledge that is invoked whenever an internal or external event occurs. Each handler within an expert system must be unique and specific for a particular event, but different expert systems may have different handlers for the same event. A handler is triggered whenever the expert is notified of an event by means of a message. The structure of a handler includes the name of the handler which is global within the expert, the event to which it can respond, an area that corresponds to the content of the incoming message, a local data area that is internal to this handler, and a block of procedural knowledge.

Similar to an event-handler, a daemon is also a portion of procedural knowledge. However, instead of waiting to be triggered by an event, the daemons are data-driven and are executed continuously one after another. Its primary objectives include monitoring the environment and behavior of the system. For example, daemons can be used to monitor an expert's problem-solving process and the states of the problem space. They can also be used to monitor the health of the processor on which the expert resides, or to monitor some external behaviors/events (such as the states of the aircraft's subsystems). Each daemon has a specific pattern that it matches. Once matched, the body (i.e., the action part) of the daemon will be executed.

Typically, a daemon sends out messages to alert or notify other expert systems of impending problems or states.

As pointed out earlier, the expert systems communicate with one another via messages. An expert may volunteer to inform others of new information, if it is valuable to the other experts. Alternatively, an expert may request the other expert systems to perform some inferences or to evaluate a piece of data. In the first case, the expert may not expect a reply from the recipient; while in the latter case, it may. An expert however would never wait for a reply because in a distributed environment, a reply may never be received due to network or processor failure.

Conceptually, the ESMS consists of three layers: the highest layer is the knowledge base of an expert system, the middle layer is the control structure which is responsible to execute the knowledge base of the expert system (i.e., event-handlers and daemons) and to maintain the event queue, and the lowest layer is the Communication Manager which accepts, maintains, and sends messages. In addition, there is a global memory that is shared by all the event-handlers and daemons within the system. This global memory can be used to exchange information and to maintain a global view of the responsible domain space. Figure 2 depicts the different components of the system.

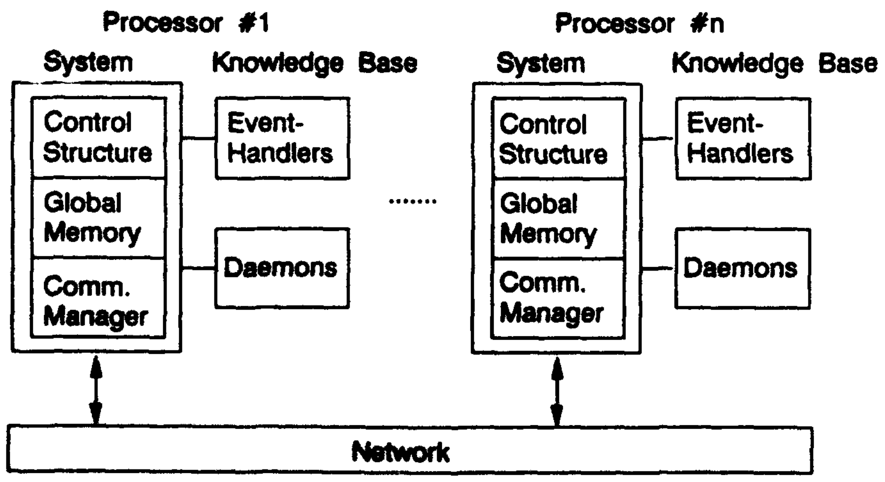


FIGURE 2. A FRAMEWORK OF THE DISTRIBUTED ESMS

Messages are sent to the expert system by name, so that the sending expert system does not need to know the location (i.e., network address) of the recipient in a distributed environment. As a result, one or more experts may coexist on the same host or reside on different hosts. This feature of host independence is particularly important for fault tolerance. For example, when a processor malfunctions due to battle damage, the control structure could reconfigure and reallocate the expert from the failed processor onto another less busy processor. This feature is critical to the success of dynamic resource allocation. In this case, when a new processor/host is added to the distributed environment, the system could utilize this new resource by reallocating an expert from a crowded processor to the newly added processor. The operation of fault tolerance and dynamic resource allocation will be described further in later sections.

Each message (or event) has a priority and certainty factor associated with it, so that when an expert is handling an event of a particular priority only an event of higher priority may interrupt it. If the expert is interrupted due to a higher-priority event, it will perform the necessary task and then resume the task previously being processed. The side-effect of this is that the interrupting event may preempt or alter the behavior of the task to be resumed. The system is not capable of handling truth-maintenance.

A message can be sent to an expert system from other experts or from within the expert itself. In the latter case, it is viewed as setting a subgoal. The content of the messages can be defined arbitrarily depending on the application, as long as there is an understanding between the sender and receiver. Typically, a message must have the name of the sender, the name of the recipient, the message type, a time stamp, a "response-to" ID number, a priority value, a certainty factor, and application-specific information. Whenever a message is received, its priority is immediately compared with the current priority of the system; if it is higher, the executing task is interrupted to handle the newly arrived event. However, if the priority is lower, it is put on a message queue according to its priority level. Within each prioritized queue, the newer message is appended to the end of the queue so that older messages are processed first. There are procedures that the user can provide to manipulate the queues if the default First-In First-Out (FIFO) strategy is inadequate.

The concurrency of ESMS is accomplished by having each expert run on its own processor. Moreover, event-handlers and daemons within each system are executed as two separate processes so that the expert can solve problems and monitor its progress simultaneously. In this sense, both interprocessor and intraprocessor concurrency have been achieved. Since the whole system is running asynchronously, each section of code in the system must be self-contained. For example, when a request-reply type action is sent, the request is written inside of a daemon or event-handler and the accept-reply is written in another. In this way there is no need for a piece of code to wait for a reply message, which may never come in a distributed system.

3.1 Fault Tolerance Operation. This is an important feature of the ESMS. The system is able to detect faults in the network or on the processor and recover. This operation involves the steps detailed in the following paragraphs.

3.1.1 Saving of the Expert System States. Periodically, the states of an expert system must be saved to a central mass storage medium. This is accomplished so that in case of a processor or network failure, the information about the expert system can be recovered and downloaded to a different processor. The flaw of this approach is that the integrity of the whole system depends on the fault tolerance of the central mass storage medium.

3.1.2 Testing of the Expert System. The testing of the three expert systems is performed by the Manager Expert System, whose primary responsibility is to monitor and test the other expert systems. The testing is done through a special type of message which is sent periodically by the Manager to the other expert systems.

3.1.3 Fault Detection. After a Test message is sent to an expert system, the Manager will wait briefly for a response. If the recipient of a Test message does not respond within a certain time frame, the Manager immediately assumes that a fault has occurred and performs a fault-recovery procedure.

3.1.4 Fault Recovery. At this point, the system checks whether any other processor/host is idle or not being fully utilized. If so, the expert system(s) from the failed processor is reallocated to this new host. If no idle processor is available, the Manager then finds the least active or crowded processor and reallocates the expert system(s) onto it.

3.2 Dynamic Resource Allocation Operation. Another important feature of the ESMS is the ability of the system to dynamically utilize new processor resources when they become available. The fault-tolerance aspect of the system also utilizes this feature in order to reconfigure and reallocate an expert system from a failed processor to a new one.

4.0 TECHNICAL ACCOMPLISHMENTS

This section covers the following subjects: ESMS distributed inference engine, domain-specific expert systems, and integration with GENASIS. The first is the detailed report on the ESMS distributed inference engine. The second describes the inner workings and accomplishments of the four cooperating expert systems, and the third details the integration of the system with the real-time flight simulator.

4.1 ESMS Distributed Inference Engine. Since the primary focus of this project was to develop a flexible distributed Artificial Intelligence inference engine, major efforts were allocated for this area. When an expert system is written under the ESMS, it can take advantage of the features that the inference engine provides. The following paragraphs contain descriptions of the major features and accomplishments in this area.

4.1.1 Processor Independence. An important feature of distributed processing is the ability to execute a program independent of the processor on which it is hosted. Under the ESMS, an expert system can cooperate and communicate with its peers independent of their location (i.e., network address). As far as the expert system is concerned, it does not matter whether its peers are running on other processors in the network or on the same processor. When an expert system needs to communicate with its peers, all it needs to do is to specify the expert system's name. An analogy of this is to mail a letter to someone with only the name of the recipient on the envelope omitting the address and zip code. Upon receipt of the letter, the post office will automatically look up the person's address and zip code, and deliver the letter. The sender does not need to be concerned with a change of address when the person moves. Of course, this approach is only possible when each system has an unique name.

The most obvious advantage of this approach is that at run time an expert system can be reallocated to a different computer while the processing and reasoning of the whole system remains intact. This is particularly important for fault-tolerant processing; when a computer fails, the expert system on that computer can be moved to another computer and continue processing from the point where it failed. This concept can be visualized in Figure 3, where Expert 1 and Expert 2 may run on a single computer or on two separate computers.

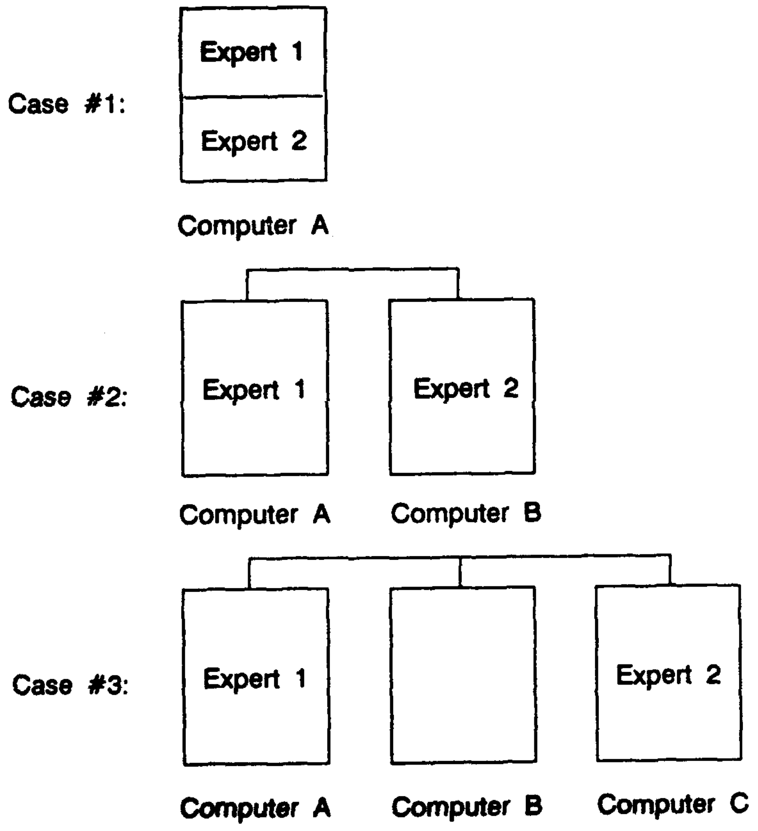


FIGURE 3. PROCESSOR INDEPENDENCE

4.1.2 <u>Message Passing</u>. In a distributed cooperating expert system environment, each expert system is responsible for its own area of expertise, but it must cooperate and exchange information with the other expert systems in order for it to behave intelligently. For example, in the avionics domain the Route Planner cannot re-plan a flight route to a new target until it consults with the Weapon Expert for weapon and chaffs/flares information. It also must consult with the Manager Expert for fuel information. The communication among the four expert systems is through messages. They can communicate with each other regardless of whether or not they all reside on the same host.

Theoretically, there are three types of messages.[2] However, the ESMS currently treats them all the same. The first type of message is a request for information (or action). This type of message will be used when an expert system requests information from another expert system or requests the expert system to perform a task. When a request is sent, the sender expects a reply from the recipient. For example, the Manager Expert System may request the Route Planner to find a new route. When it finds a solution, the Route Planner then sends the result to the Manager. The second type of message is when an expert system voluntarily informs other expert systems of some interesting information or conclusion it has reached. In this case, the sender does not expect a reply from the recipient. The third type of message is a reply. This message is in response to a request; therefore, the sender of a reply message does not expect a response from the recipient.

When an expert system makes a request of its peers, it does not wait for a reply before it continues processing. This is the implementation chosen for this distributed environment because the network and/or the processor may fail and a reply may never arrive. In the ESMS, the sending of a request and the receiving of a reply are in two separate pieces of self-contained code. Again, using the post office as an analogy, a person does not wait in the post office for a reply after sending a letter.

In order to trace and keep track of the problem-solving process of each expert system, a message also implicitly possesses some context-sensitive information[3] (i.e., an expert system's problem-solving states). This information will be implicitly incorporated in the outgoing message based on

---

[2] In this document, sometimes message is interchangeable with event.

[3] For more information on context-sensitive problem-solving, please refer to 4.1.6.

the sender's current context. For example, under the context of GOING-TO-A-TARGET, the Manager Expert requests the Route Planner to plan a new route. The behavior of the Route Planner would be very different if the context is COMING-BACK-FROM-A-MISSION. This information is critical for an expert system's fault-recovery, which is described later.

In a fault-tolerant system, a failed expert system can be recovered if its problem-solving states prior to the failure are recoverable. In the ESMS, an expert system's problem solving states are its outgoing and incoming messages which must be saved periodically. When an expert system is recovered from a failed processor, the saved outgoing messages are used by the expert to recall what it is expecting to receive from the other expert systems, while the incoming messages are used to determine what other expert systems are expecting. When the messages are recovered, only the relevant ones under the appropriate context(s) are used and the rest are discarded.

4.1.3 Event-Driven and Data-Driven Inference. When the Manager Expert System is alerted of an aircraft malfunction, it determines that it should request the Route Planner to find an emergency landing site. The action taken by the Manager Expert is called event-driven because it is triggered by an event which occurred. An event can be a conclusion reached by another expert system or a subgoal set by the expert system itself. Detecting a malfunction is a data-driven process because a system is continuously monitoring the aircraft states for abnormal readings. The aircraft states and similar information are pieces of data which may change over time. Depending on the changes in the data, the system must respond appropriately.

In the ESMS, event handling is encoded in an event-handler, and data monitoring is encoded in a daemon. Both of these inferences may occur concurrently. For example, an expert system may simultaneously respond to an event using an event-handler and monitor other data using a daemon. An advantage of having concurrency is that an expert system may use a daemon to monitor the problem-solving progress of an event-handler.

Since each event may have a different urgency or priority, it is important that the system be capable of responding accordingly. An event may have one of four priority levels: emergency, top-priority, high-priority, and normal. The meaning of the priority levels is self-explanatory. The handling of different priority events is described in the section on Interruptible Inference.

The execution of daemons is governed by their context. In other words, not all daemons may be active at once. Only the daemons under the appropriate context may execute, and the daemons under the Universal-Context will execute under all situations. The details on the context-sensitive processing is provided in the section on Context-Sensitive Problem Solving. In short, context is used to provide an efficient way for an expert system to focus its problem-solving process. For example, there are daemons to monitor and respond to problems under the context of Take-off, and different daemons for the context Combat. Figure 4 shows the flow of the system problem solving as well as the relationships between event-handlers and daemons.

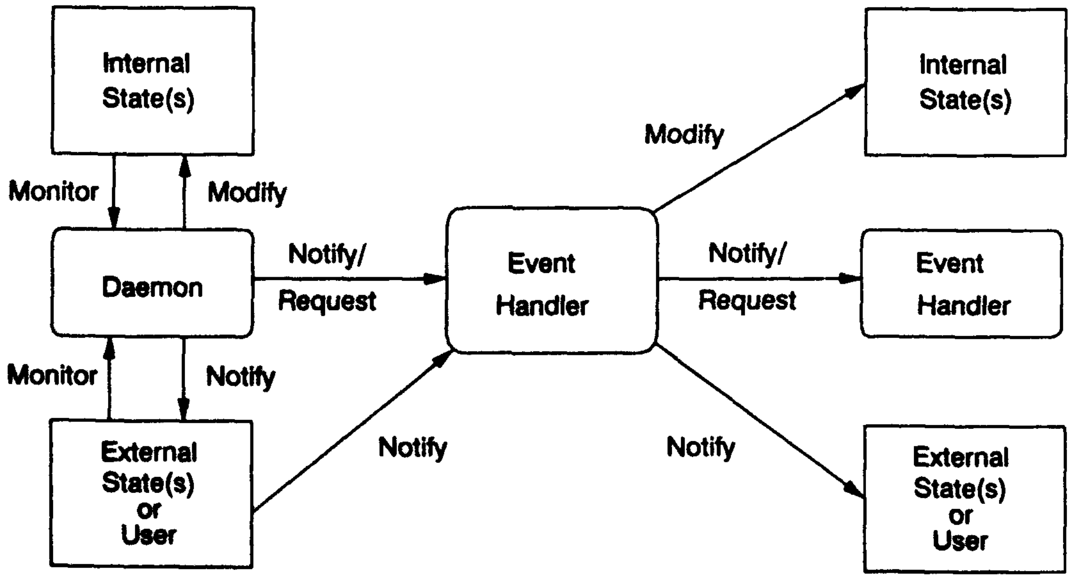


FIGURE 4. RELATIONSHIP BETWEEN DAEMONS AND EVENT-HANDLERS

4.1.4 Interruptible Inference. A shortcoming of most traditional inference engines is that they are incapable of handling interrupts. One reason is that they were originally developed for non-real-time applications involving human interaction. In that case, there is no interrupt. However, in a real-time domain such as flight control and pilot aiding, asynchronous events with different priorities may occur. Therefore, when the expert system is resolving an event and a different event of higher priority occurs, the expert temporarily suspends its present task and responds to the higher priority event. Upon completion of the latter task, the expert then resumes the interrupted one. For example, the Route Planner is in the process of re-planning a route based on the available information, and suddenly it is alerted of a nearby pop-up threat. The Route Planner must then preempt the re-planning and utilize the new information for another re-planning. In the case of a lower priority event, it will be appended to the rear of a message queue based on the event's priority level.

The ESMS is capable of handling interrupts because inference engines for event-handlers and daemons are implemented using processes. As in most computers, the processes on a *Lisp machine* are interruptible and so the inference is interruptible. However, a drawback of using processes on the Lisp machine is that they are not efficient. When there is a lot of context switching[4] between multiple processes, a noticeable degradation in performance may occur.

It has also been observed that the use of certain process functions may produce unpredictable behavior. For example, when an expert system has no message/event to execute, its event-handler process will be arrested[5] or suspended until the expert system is notified of a new event. At that time, the event-handler process will be enabled again by the control structure. For

[4] Context switching occurs when the operating system (OS) temporarily suspends a process and saves all the state information about the process. Once this is completed, the OS starts another process with a different set of state information.

[5] Arresting of a process means that it is forbidden to run. Therefore, it becomes inactive until it is given a run reason.

some reason, it seems that when the process is enabled, it is not scheduled or allowed to run. As a result, the expert system remains inactive. In some instances, the process is suddenly allowed to run when it receives a second message, and it immediately handles both pending events.

It is strongly believed that interruptible inference is critical to any real-time expert system application; therefore, this is implemented in the ESMS.

4.1.5 Shared Global Variables. There are two primary reasons that the shared global variable feature is included in the ESMS. The first is to improve the efficiency of information sharing among cooperating expert systems being run on multiple processors. In the domain of pilot aiding, mission information and aircraft states are some of the information that can be shared among multiple expert systems. For example, aircraft state information is needed by three of the expert systems. When the Route Planner needs to know about the fuel quantity, it explicitly requests this data from the Manager Expert by sending a message. However, this approach is very inefficient. Alternatively, whenever the fuel quantity information is changed, it is automatically updated locally within the Route Planner (as in the other two expert systems). Therefore by the time the Route Planner tries to use this data, it should have the latest value locally. (See Figure 5)

The second reason for using shared global variable is to test the real-time issue of a software implementation of this feature (as opposed to a hardware implementation). The performance of shared variables with the software approach is more than adequate when there is limited processing on the processors that share global variables. However, when the expert systems are busy problem solving, the throughput of employing shared global values drops off sharply. For this reason, it seems that a hardware implementation may be needed.

The testing of the real-time response of global variable sharing is performed rather unscientifically. It is based on observation by the naked eye instead of computer tabulation. For example, when two processors devote all resources to perform global variable sharing, the setting of a new value on one processor can be accessed instantaneously on the other processor. However, when there is other processing on one or both of these processors, one can observe a noticeable degradation in performance. This observation makes sense, but such degradation may prove intolerable.

The declaration and use of global variables is similar to that of Lisp variables. To declare a global variable, use DEFGLOBALVAR (as opposed to DEFVAR). To set, use GLOBAL-SETQ (as opposed to SETQ). They are accessed in the same manner as Lisp variables.

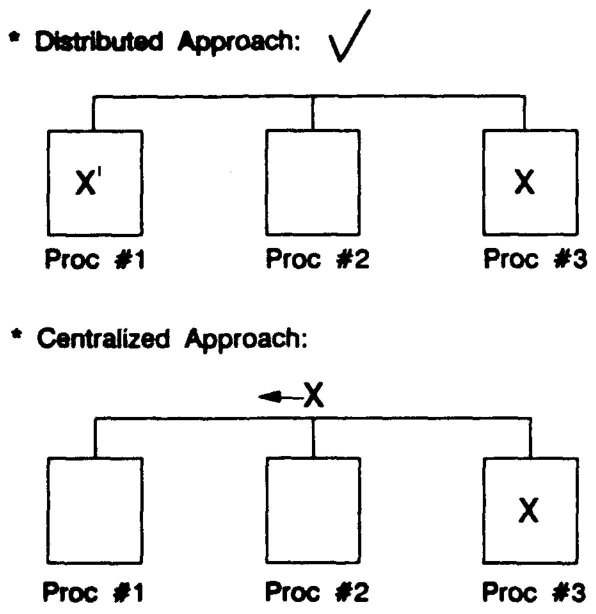


FIGURE 5. SHARED GLOBAL VARIABLES

4.1.6 Context-Sensitive Problem Solving. As pointed out briefly in the section on event-driven and data-driven inference, context-sensitive problem solving is used to improve efficiency and to focus problem solving. It helps to improve system throughput with contexts since the inference engine does not waste its resources searching for solutions in the irrelevant problem space. Presently, only the data-driven inference (i.e., daemons) is run using contexts. The contexts for event-driven inference (i.e., event-handlers) could be developed in the future.

In the ESMS, context is used to group together related daemons for a particular type of problem solving. Therefore, when a context is active, only the daemons in that context become active. This is in contrast with traditional rule-based systems such as OPS5 where all rules can potentially be fired at any one time. When a limited set of daemons is active, there is less overhead and the system becomes more efficient. This is best described by the following example.

In the avionic domain, there are many different contexts, i.e., Taxi, Take-Off, Cruise, Combat, etc. When the system is in the middle of the Combat Context, only the daemons relevant to this context can potentially be fired. It does not make sense to check the daemons under the Taxi Context or the Take-Off Context until either of these contexts becomes active. For each context, there is a metarule associated with it. This metarule is used by the control structure to activate the appropriate context when the conditions are met. Therefore, all context switching is done automatically by the control structure. At any one time, more than one context may be active.

Context is also useful for fault recovery. For instance, when an expert system is in the middle of solving a problem under one context and the processor fails, the expert system upon recovering from this failure may not want to resume the previous task because the context has changed and that task is no longer relevant. In order to implement such a fault-tolerant system, a global context common to all expert systems seemed to be necessary. This means that in addition to the local contexts specific to an individual expert system there needed to be some contexts that are shared by all expert systems so that they all have the same view of the world.

4.1.7 Fault Tolerance. The execution of an expert system is host independent as described earlier, so that when a processor fails the expert system can be reallocated and run on another processor. This feature was demonstrated successfully in the Avionics Laboratory at Wright-Patterson Air Force Base.

The approach used is that an expert system must periodically save some of its state information (i.e., incoming and outgoing messages/events). This saving of information is done automatically on a set time interval as specified by the user. Each message must have an implicit context associated with it, so that when a failed expert system is recovered, only the messages under the relevant context is restored. The expert system may resume its problem solving process based on the recovered incoming and outgoing messages and on the global context described earlier.

It is the responsibility of the Manager Expert to check on the health of the expert systems periodically. This is accomplished by sending the expert systems test messages. When an expert does not respond within a certain time frame or when the message cannot be sent, the Manager assumes that the processor or expert has failed. The Manager then reloads the knowledge bases and data bases of this expert onto another less busy computer, and re-establishes the communication links with the rest of the system. Once an expert system is recovered, it should be able to produce the same output as before if it is given the same set of input. However, since more than one expert system is now running on the same processor, the throughput of the expert systems on that processor will degrade. Such graceful degradation is important for most distributed AI applications.

The assumption of this approach is that when a processor fails, the knowledge bases and data bases of an expert system can be recovered from a central mass storage medium. This assumption is in-line with the PAVE PILLAR architecture where a central file server is used for all disk I/O.

Key design decisions for this fault tolerance feature include

a. the types of information/data which must be saved and which are pertinent to the recovery and consistency of an expert system. There are two types of information: information that is specific to a particular domain, and information that is about the state of an expert system. For example, information specific to the route planning domain includes current route, next waypoint, threats, and the map display. The information about the state of the Route Planner includes its event queue, current event, and monitor list.

b. the frequency of saving the information. For example, the developer could specify that once every minute the states of the Weapon Expert System should be saved. It may be the case that the states of the Route Planner Expert System should be saved more often because they change more often. Presently, the states of the expert systems are saved once every minute.

c. the frequency that the state of an expert system and its processor (i.e., TI Explorer) must be tested by the Manager Expert System in order to detect and recover from any failures. Presently, each expert system is tested once every second. Again, this value can be changed when necessary.

4.2 Domain Specific Expert Systems. To demonstrate the validity of the ESMS architecture, a set of four cooperating expert systems for pilot aiding has been developed. Each of these experts is responsible for its own area of expertise, but they also must communicate and cooperate with each other in order to behave intelligently. It has been demonstrated that these expert systems do indeed cooperate with each other, whether they all run on a single computer or on multiple computers.

Since the primary focus of this project is to develop the distributed AI inference engine, a smaller amount of effort has been devoted to developing the knowledge bases of these expert systems. As a result, they may lack the depth of truly intelligent systems. In any case, a detailed description of each of the expert systems is presented.

4.2.1 Manager Expert System. The responsibility of the Manager Expert System is to oversee the operation of the other three expert systems and to ensure that the needs of the pilot are met. It also monitors the status of the mission progress and development. When there is a change of mission, it immediately requests that the Route Planner re-plan for a new flight route.

When a pop-up threat occurs, it alerts the pilot of the situation as well as the Weapon Expert and the Route Planner. In this case, the Weapon Expert may discharge chaffs and flares to confuse the threat, and the Route Planner may plan a quick escape route to avoid the threat.

Periodically, the Manager sends out messages to check the status of the three expert systems. If an expert does not respond within a certain time frame, it assumes the expert has failed and the Manager will try to re-configure the system to alleviate this failure.

4.2.2 Route Planner Expert System. The present state of this knowledge base allows the Route Planner to plan and re-plan flight routes to avoid known and pop-up threats. Prior to takeoff, when the Route Planner is given the threat situation and the mission statement (i.e., set of waypoints and targets), it plans a route to the targets and back to a friendly base avoiding all the threats. However, when enroute to the target, unexpected threats may be detected. Depending on the location of the threat relative to the aircraft, the Route Planner must determine quickly a new route in order to avoid the threat and to ensure maximum safety.

There are four cases for handling unexpected pop-up threats:

a. The Route Planner is alerted of a pop-up threat, but the threat is not along the path of the planned route, so it is ignored.

b. The threat pops up along the route but is further down the flight path and far enough away from the current position of the aircraft, so the Route Planner can take its time to re-plan a new route around the threat.

c. The threat is along the route and the aircraft is just outside of the threat envelope; therefore, the Route Planner must quickly re-plan to avoid entering it.

d. An unexpected threat pops up near the aircraft and its envelope encompasses the aircraft; the Route Planner must quickly determine whether it is worthwhile to escape outside of the threat or to continue its original course. In either case, chaffs and flares must be dispensed to confuse the threat.

In all of the above situations, the Route Planner will try its best to avoid the threat. Only when it fails to find a new route or when the aircraft is already too deep inside the threat envelope does it become necessary for the aircraft to continue its original course and fly inside the threat envelope. Many heuristics are still needed in order for the expert to handle other situations.

In addition to handling threat avoidance during the flight, the pilot or the Manager Expert may decide to use an alternate mission. When notified of a mission change, the Route Planner must be able to re-plan quickly using the new mission statement.

4.2.3 Weapon Expert System. For this expert system, there is a display showing the outline of an aircraft with all of its weapons/missiles and its self-defense countermeasures such as chaffs and flares. When the aircraft unexpectedly enters a threat envelope, this expert system may be instructed to dispense some of its chaffs. The dispensing of the chaffs is displayed on the output window in real time.

The Weapon Expert is also responsible for matching and allocating weapons against threats. For example, prior to entering a threat, the Weapon Expert may suggest firing some of its missiles to destroy the threat so that it is unnecessary for the Route Planner to re-plan around the threat. When the aircraft enters the target area, it will suggest dropping its bombs to destroy the target. Again, the firing of weapons and missiles is displayed on the output window.

4.2.4 Situation Awareness and Display Expert System. When presented with two or more alerts, the SADE first prioritizes the alerts prior to presenting them to the pilot. Therefore, the pilot is not overwhelmed and confused by multiple alerts. For example, when flame-out and fuel-leak alerts occur simultaneously,[6] the SADE System should alert and advise the pilot to take care of the flame-out first, prior to presenting the second problem. Currently, all of the output from this expert system is text. A synthesized voice (i.e., DECTalk) for this function was demonstrated but with limited usefulness. The first limitation of the synthesized voice is that it was very unhumanlike. Perhaps a digitized voice may be more realistic. The second limitation is that the throughput of the system (DECTalk) was inadequate. This is partially due to the slow baud rate (2400) that the DECTalk accepts from the RS232 port. In any case, the digitized voice and graphics display for pilot aiding needs to be further investigated.

4.3 Integration with GENASIS. Prior to integrating with the GENASIS flight simulator, the four cooperating expert systems were first linked to the NavModel simulator enabling each of these systems to run on four TI Explorers and to request and receive flight data from the NavModel. For example, on the Route Planner output window one can visualize the flight of a simulated aircraft with its position, heading, and other related information while a pseudo pilot is controlling the aircraft from the NavModel simulator on a MicroVAX. For the purpose of this application, only the Route Planner Expert System and the SADE System request aircraft data from the simulator. The frequency of the requests can be set dynamically before each run of the expert system. For example, the Route Planner Expert System is currently requesting data every 1/3 of a second.

The interface development between the expert systems (i.e., TI Explorers) and the GENASIS flight simulator took place toward the end of the project. By November 1989, the ESMS was able to request and receive real-time flight data from the GENASIS simulator. In addition to aircraft state information, the

[6] In this example, flame-out and fuel-leak are two separate and unrelated problems. When the fuel-leak causes the fire, the fuel-leak problem should be addressed first.

Manager Expert System was also able to request mission information from the simulator. The interface between the TI Explorers and the simulator was synchronous in nature. In other words, an expert system must make a request in order to receive any data. In the following paragraphs, a detailed account of the final demonstration of the ESMS system is presented.

Prior to the demonstration of the system, UFA and Air Force personnel met and discussed the accomplishments and future directions of this project. The participants agreed that this research benefited and tremendously interested the Air Force, particularly the Fault Tolerance feature of the system.

For this demonstration, each of the four expert systems (i.e., Manager Expert System, Route Planner Expert System, Weapon Expert System, Situation Awareness and Display Expert System) reside on their own TI Explorer Lisp machine. The GENASIS cockpit was controlled by a pilot. When the whole system was brought up, the Manager Expert System first requested a mission from the cockpit. It in turn requested the Route Planner Expert System to plan a route to and from the target(s) avoiding all threats based on this mission. After a route was found, the Route Planner then sent the route to the Manager. Upon receiving this information, the Manager notified the Situation Awareness and Display Expert System, so that it could monitor the target(s) and any pop-up threats.

At this point, the Route Planner Expert System started requesting aircraft state information from the GENASIS cockpit. The display of the aircraft could be visualized on the output window of the Route Planner. (See Figure 6) By now the Situation Awareness and Display Expert System also requested aircraft data from the cockpit. The output format of this expert system is similar to the one shown in Figure 7. The output of the Weapon Expert System is similar to the one displayed in Figure 8.

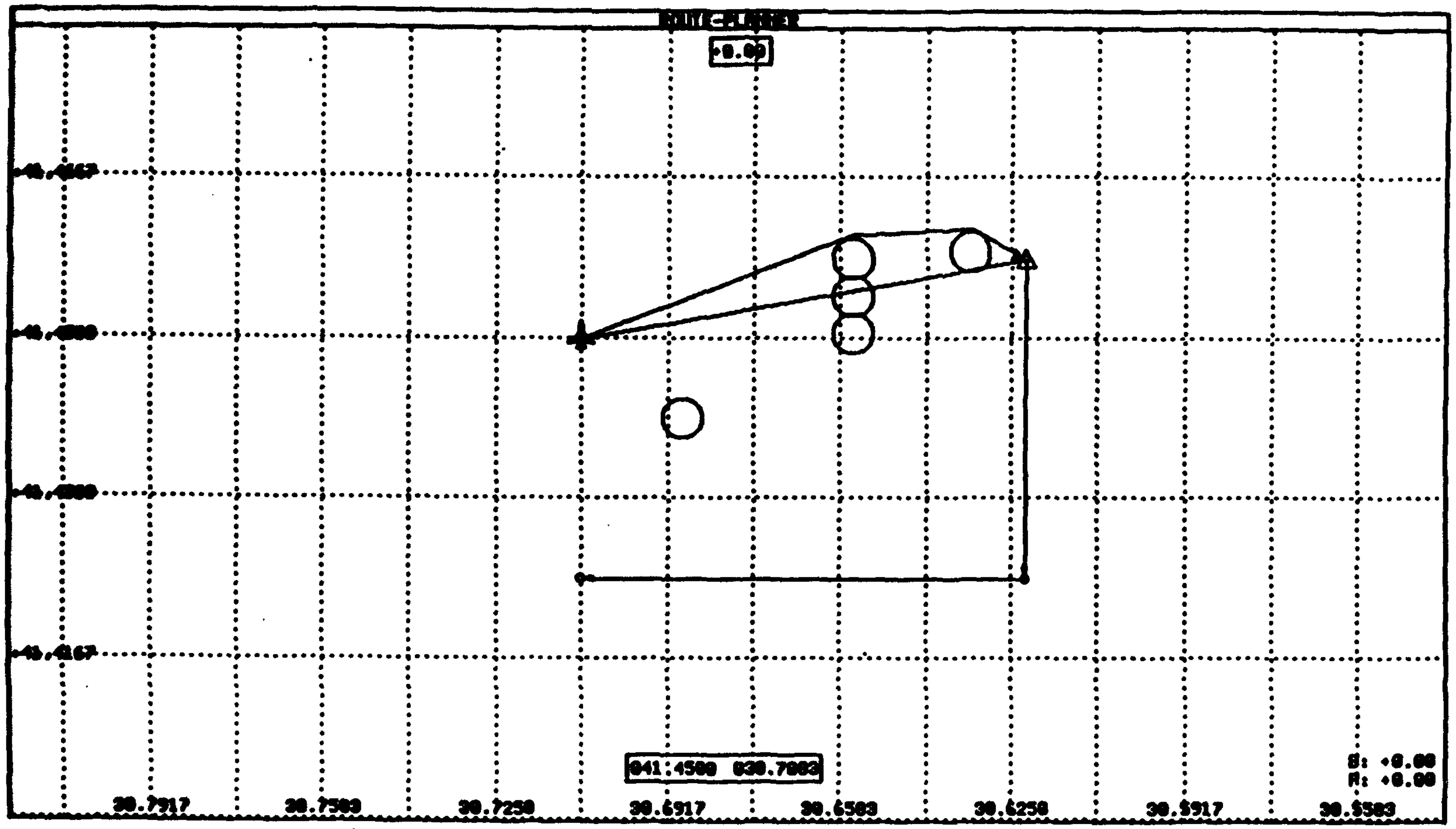
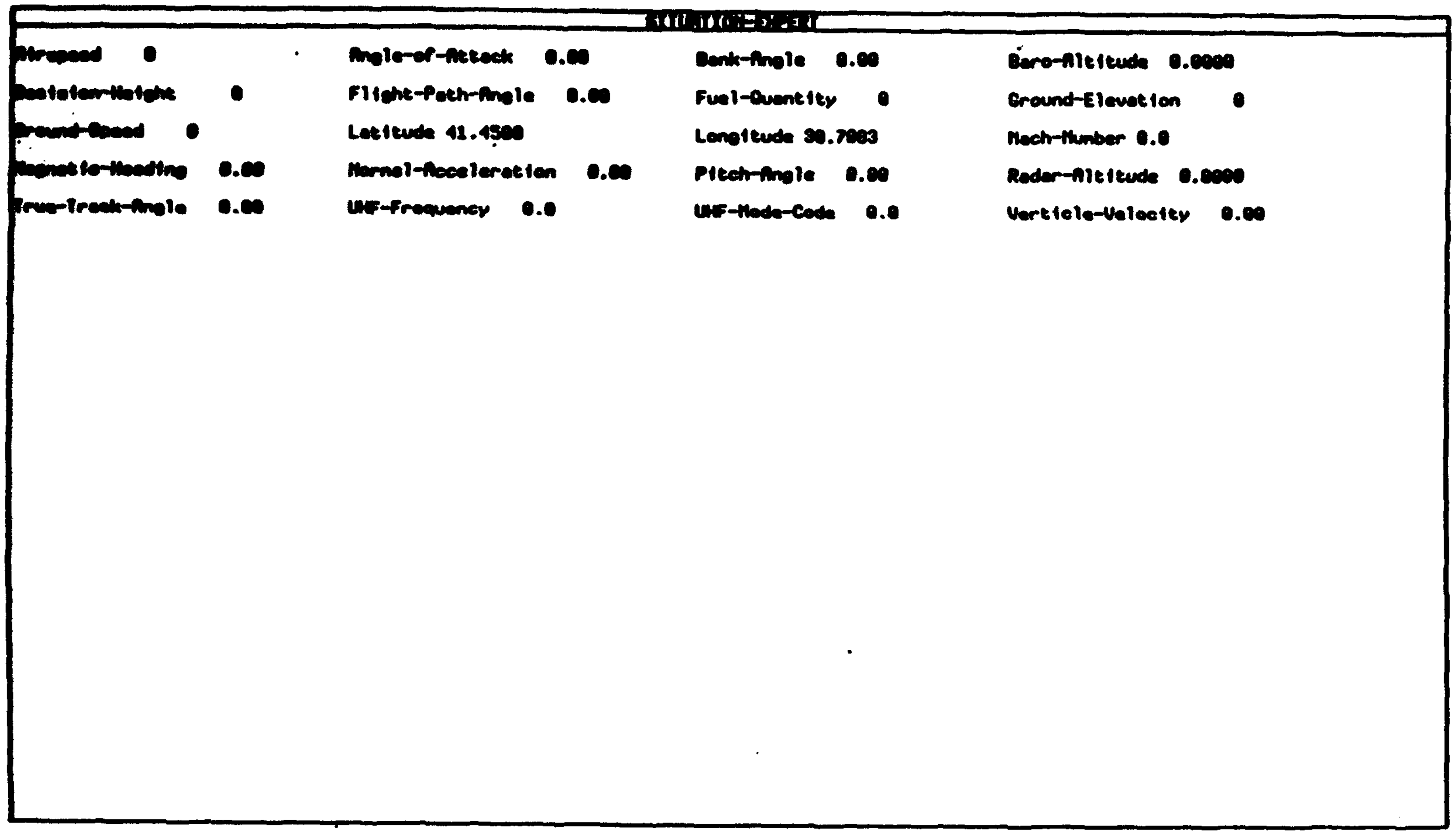


FIGURE 6. OUTPUT WINDOW OF THE ROUTE PLANNER



```
                                    SITUATION-EXPERT
Airspeed    0            Angle-of-Attack   0.00      Bank-Angle   0.00      Baro-Altitude  0.0000
Decision-Height    0     Flight-Path-Angle   0.00    Fuel-Quantity    0     Ground-Elevation    0
Ground-Speed    0        Latitude 41.4500            Longitude 30.7083      Mach-Number 0.0
Magnetic-Heading   0.00  Normal-Acceleration   0.00  Pitch-Angle   0.00     Radar-Altitude  0.0000
True-Track-Angle   0.00  UHF-Frequency   0.0         UHF-Mode-Code   0.0    Verticle-Velocity   0.00
```

FIGURE 7. SADE SYSTEM OUTPUT FORMAT

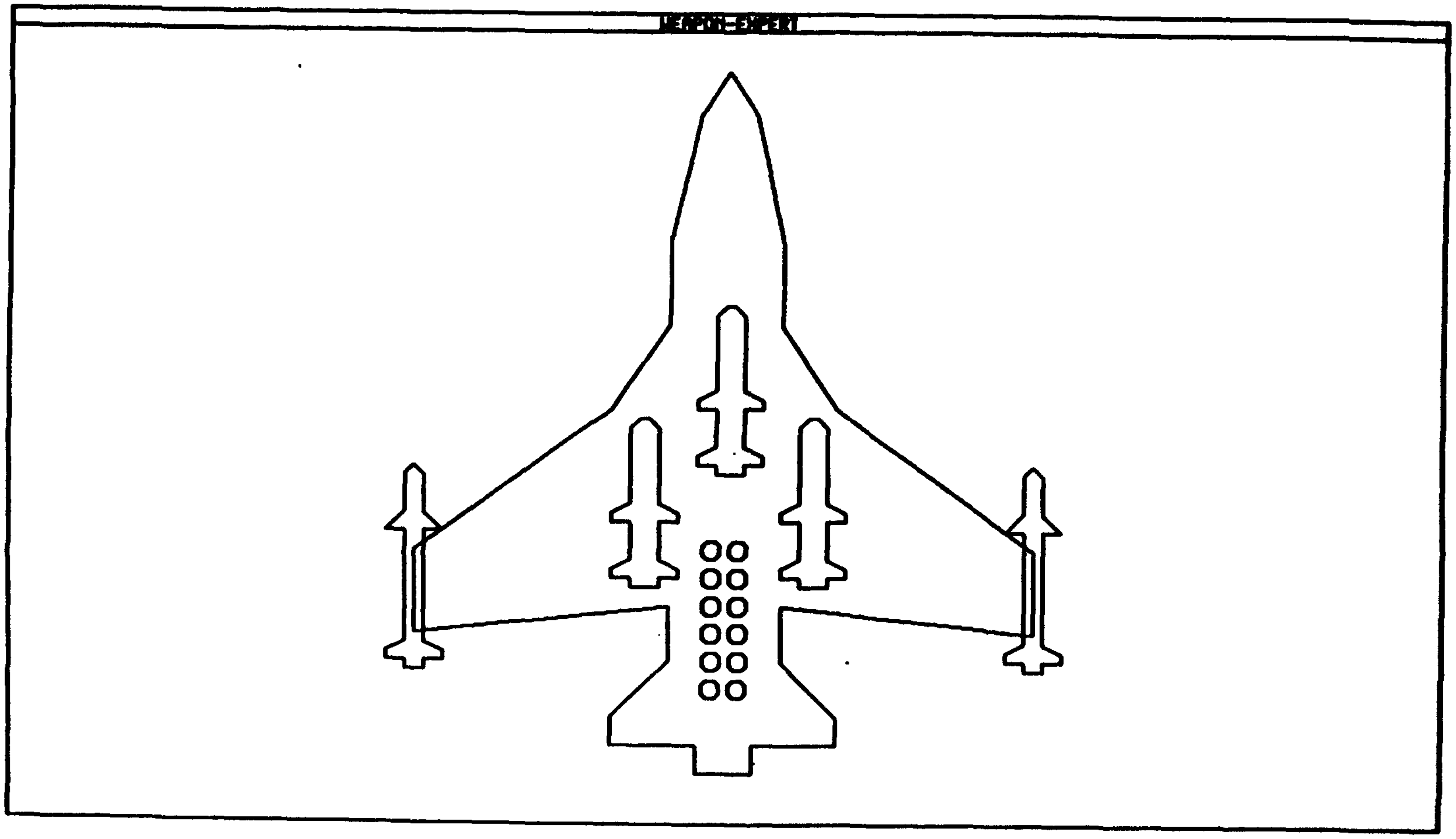


FIGURE 8. OUTPUT OF THE WEAPON EXPERT SYSTEM

While the aircraft was flying, a pop-up threat was added to the system manually from the keyboard. This threat was detected and the Route Planner was alerted to plan an alternate route avoiding the threat. For the purpose of the demonstration, the envelope of this threat was shown to encompass the aircraft. As a result, the Situation Awareness and Display Expert System detected this danger and alerted the Weapon Expert System to discharge chaffs and flares. It could be seen on the output window of the Weapon Expert System that a number of chaffs were dispensed. Next, the Route Planner concurrently re-planned and found a route outside of the threat, which was shown on the output window of the Route Planner. It can be noted that each of these expert systems solved problems independently and in parallel. Therefore, a problem could be resolved quickly. This is the advantage of distributed AI.

At this point, one of the processors on which the Weapon Expert System resided was shutdown unexpectedly to simulate processor failure due to battle damage. The Manager Expert System quickly detected this failure and immediately determined onto which available processor to reallocate the Weapon Expert System. It then reloaded the knowledge base and the saved states of this expert system onto the new processor. Upon reloading the knowledge base, the Weapon Expert System was initialized to run based upon its previously saved states. The whole distributed system was restored to its previous capabilities (i.e., all four expert systems were again interacting). However, the systems were running on three processors instead of four. This exercise demonstrated the fault-tolerance feature of the system.

After a minute or so, the "failed" processor was manually rebooted. Within seconds, the Manager Expert System detected this re-booted machine. It then directed this new processor to load in the ESMS inference engine, the knowledge base, and other peripheral information of the Weapon Expert System. When completed, all four of the expert systems were running on four separate processors. Therefore, the integrity of the distributed systems was maintained and all of its computer resource was restored (i.e., all four processors were running concurrently). This exercise demonstrated the dynamic resource allocation of the ESMS.

While the above activities were taking place, the aircraft continued to fly. As viewed on the Route Planner output window, the aircraft was approaching its target which was specified in the original mission statement. When it finally closed in on the target, the Situation Awareness and Display Expert System alerted the Weapon Expert System to fire its weapons. The firing of the weapons could be visualized on the Weapon Expert System's output window. After the target was destroyed, the aircraft continued on its course and followed its flight path back to base. This ended the demonstration.

5.0 CONCLUSIONS

5.1 Real-Time AI Applications. **Use of processes is important for real-time AI applications.** For most traditional real-time applications such as process control and avionics, the ability to preempt a less important task/process is critical to the success of the whole system. Therefore, a real-time AI software must also be able to behave in a similar fashion. Similar to traditional real-time systems, the use of processes will lend itself nicely to such control. In the ESMS, each expert system has its internal state a priority. The higher priority process will run first. If an impending event has an even higher priority, the expert system will temporarily suspend the current task and take care of the impending event. Upon the completion of the interrupting event, it then resumes the previous task. With the use of processes, the internal states of the process will be automatically saved when it is interrupted. These states will be restored when the process resumes.

5.2 Real-Time Processing. **Lisp machines are not fast enough for real-time processing.** Actually, the term realtime must be qualified here. It is referring to any processing, such as avionics, that requires a response time of a second or less. Lisp machines are not considered fast enough for such applications due to the following two reasons: First, as pointed out in 1., processes are important for real-time AI applications. Lisp machines do not handle processes very well, particularly when there are multiple processes and the system must interrupt and swap among these processes. Second, due to the non-deterministic nature of the Lisp language itself, the system cannot be depended on for things to happen at exactly the right moment.

5.3 Lisp Language. **Lisp language is still good for rapid prototyping and initial development.** Due to the interactive nature of the language, Lisp is still good for rapid prototyping and initial development. Therefore, it is relatively quick and simple to incorporate new changes to the system, i.e., new changes can be loaded in while the system is still running. The new changes will be immediately put into effect without disrupting the rest of the system. In other languages, such as C, when a part of the program is changed, that part must be re-compiled and re-linked with the rest of the program.

Such a task can be time-consuming. Another benefit of the Lisp language is that it is excellent for symbolic programming, allowing the developer to conceptualize the problem at a higher level.

5.4 Distributed Expert System Execution. **Distributed expert system execution is difficult to trace in contrast to a traditional system where a single thread of logic is followed by the system, i.e., execution is sequential in nature.** Here there are multiple threads and each is acting independently of each other. Therefore, at any one time, more than one thing could occur and sometimes it is difficult for the developer/user to understand exactly what sequence of events has taken place in the system.

5.5 Separation of Knowledge Bases/ESMS Inference Engine. **Separation of the expert system knowledge bases and the ESMS inference engine is critical to its success.** Since both the knowledge bases and the inference engine undergo continual revisions and changes, their separation has made the process of enhancing the system much easier. For example, if there is a change to the knowledge base, it can be done without disturbing the inference engine and vice versa. Furthermore, the fact that the two entities are separate has made the overall size of the program much smaller and well modularized.

5.6 System Development. **System development must be done with close user interaction and feedback.** This is true for all software development projects. In this case however, one of the UFA engineers spent many months at the Avionics Laboratory defining the needs and requirements of the Air Force. Such close interaction is particularly critical when the interface had to be worked out between the ESMS and the flight simulator. The fact that an engineer from UFA was working in the laboratory side by side with the engineers from the Air Force eliminated any communication barrier that may otherwise have interfered with progress.

5.7 Real-Time Distributed Fault-tolerant AI System. **Implementing a real-time distributed fault-tolerant Artificial Intelligent system is a difficult problem.**

6.0 RECOMMENDATIONS

This project demonstrated how it is possible to incorporate expert systems to reduce pilot workload and increase effectiveness. The simulation developed showed how a pilot would be able to concentrate on the more important tasks while the system handles routine tasks.

A recommendation for future research is that further testing of the ESMS occur with the focus toward fault-tolerant problem-solving capabilities. It is further recommended that testing and fine tuning of the fault-tolerant expert system take place. Further research is needed on the distributed expert system and the use of processes which is very important in any real-time expert system.

7.0 BIBLIOGRAPHY

[1] Berning, S., D.P. Glasson and J.A. Guffey, "Adaptive Tactical Navigation Concepts," IEEE National Aerospace Electronics, (1986) Vol 4, pp. 1235-1242.

[2] Bond, Alan H., and Les Gasser, "Readings in Distributed Artificial Intelligence," San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1988.

[3] Buchanan, B.G., and E.H. Shortliffe, Rule-Based Expert Systems, Reading, MA: Addison Wesley, 1984.

[4] Davis, R., "Report on the Workshop on Distributed AI," SIGART Newsletter, (October 1980) No. 73, pp. 42-52.

[5] Davis, R., and R.D. Smith, "Negotiation as a Metaphor for Distributed Problem-Solving," Artificial Intelligence, (January 1983) Vol 20, No. 1, pp. 63-109.

[6] Durfee, E., and V.R. Lesser, "Incremental Planning to Control a Blackboard-Based Problem-Solver," Proceedings of the Fifth National Conference on Artificial Intelligence, August 1986, pp. 58-64.

[7] Durfee, E., V.R. Lesser, and D.D. Corkill, "Cooperation Through Communication in a Distributed Problem Solving Network," Distributed Artificial Intelligence, California: Morgan Kaufmann, 1986, pp. 29-58.

[8] Erman, L., F. Hayes-Roth, V.R. Lesser, and R.D. Reddy, "The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," ACM computing Surveys, (1980) Vol 12, pp. 213-253.

[9] Gasser, L., C. Braganza and N. Herman, "MACE: A Flexible Testbed for Distributed AI Research," Distributed Artificial Intelligence, 1987, California: Morgan Kaufmann, pp. 119-152.

[10] Green, P.E., "AF: A Framework for Real-Time Distributed Cooperative Problem-Solving," Distributed Artificial Intelligence, California: Morgan Kaufmann, 1987, pp. 153-175.

[11] Green, P.E., "Resource Control in a Real Time Target Tracking Process," Proceedings of Fifteenth Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, November 1981, pp. 424-428.

[12] Hsu, C.C., S.M. Wu and J.J. Wu, "A Distributed Approach for Inferring Production System," Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 1987, pp. 62-67.

[13] Lesser, V.R. and D.D. Corkill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks, AI Magazine, (Fall 1983) Vol 4, No. 3, pp. 15-33.

[14] McArthur, D., R. Steeb, and S. Cammarata, "A Framework for Distributed Problem-Solving,"Proceedings of the National Conference on Artificial Intelligence, Pittsburg, PA, 1982, pp. 181-184.

[15] Nelson, Victor P., and Bill D. Carroll, "Tutorial: Fault Tolerant Computing," Washington, D.C.: IEEE Computer Society Press, Computer Society Order Number 677.

[16] Pohlmann, L.D., and J.R. Payne, "Pilot's Associate Demonstration One: A Look Back and Ahead," IEEE National Aerospace Electronics, (1986) Vol 4, pp. 1176-1183.

[17] Schira, J.A., Jr., "Fighter Pilots Aid by Expert Systems," Conference on Intelligent Systems and Machine, 1984, pp. 364-369.

[18] Shartz, Sol M., and Jia-Ping Wang, "Tutorial: Distributed Software Engineering," Washington, D.C.: IEEE Computer Society Press, Computer Society Order Number 856.

[19] Shelnutt, J.B., R.O. Stenerson, P.C. Nelson, and P.S. Marks, "Pilot's Associate Demonstration One: A Look Inside," IEEE National Aerospace Electronics, (1986) Vol 4, pp. 1184-1189.

[20] Skillman, T.L., Jr., "Distributed Cooperating Processes in a Mobile Robot Control System," Workshop on Blackboard Systems for Robot Perception and Control, June 1986.

[21] Smith, R.G., "Report on the 1984 Distributed Artificial Intelligence Workshop," The AI Magazine, (Fall 1985) pp. 234-243.

[22] Stankovic, John A., and Krithi Ramamritham, "Tutorial: Har Real-Time Systems," Washington, D.C.: IEEE Computer Society Press, Computer Society Order Number 819.

[23] Tenney, R.R., and N.R. Sandell, "Strategies for Distributed Decisionmaking," IEEE Transactions on Systems, Man, and Cybernetics, (August 1981) Vol SMC-11, No. 8, pp. 527-538.

[24] Weinreb, D., and D. Moon, "Flavors: Messages Passing in the Lisp Machine," A.I. Memo No. 602, MIT AI Laboratory, Cambridge, MA, November 1980.