AN EXPERT SYSTEM FOR AIRBORNE OBJECT ANALYSIS

by

DAVID EDWARD POPE

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF

MASTER OF SCIENCE
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

and

BACHELOR OF SCIENCE
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
January, 1985

Signature of Author _____
Department of Electrical Engineering
and Computer Science
January, 1985

Certified by _____
Antonio L. Elias
Thesis Supervisor

Certified by _____
Charles P. Dolan
Hughes Aircraft Company Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

AN EXPERT SYSTEM

FOR AIRBORNE OBJECT ANALYSIS

by

DAVID EDWARD POPE

Submitted to the Department of Electrical Engineering and Computer Science on January 28, 1985 in partial fulfillment of the requirements for the Degrees of Master of Science in Electrical Engineering and Computer Science and Bachelor of Science in Electrical Engineering and Computer Science

ABSTRACT

Two expert systems were designed and implemented for the purpose of performing airborne object analysis, the task of identifying as completely as possible opposing military planes on the basis of returned sensor evidence.

Knowledge engineering on airplane characteristics, sensor capabilities, and pilot protocols was performed, and a base of sixty rules, including mission-level, formation-level, and target inferencing, was developed.

The first expert system, designated OAS1, was rule-based and served primarily as a demonstration, feasabilty, and learning tool. A Bayesian inference net was integrated into the system to evaluate belief maintenance requirements.

The second expert system, designated OAS2, is demon-based, represents formation and airborne object knowledge in frames, and uses procedural attachments to control the spawning of demons. Performance on a six-object scenario with six possible identities approaches real-time, using LISP on engineering workstations.

Thesis Supervisor: Dr. Antonio L. Elias
Assistant Professor of
Aeronautics and Astronautics

TABLE OF CONTENTS

APPENDIX B

LIST OF FIGURES

Acknowledgements

This thesis could not have been completed without the help of a great many people at home, at the Massachusetts Institute of Technology, and at Hughes Aircraft Company, where I completed this research. It is a pleasure to acknowledge them each in turn.

Sue Baumgarten and Brock Fifer, my senior-level managers at Hughes, took an interest from the beginning in my search for a thesis topic, and worked hard to find the best location for me at Hughes. They provided support and encouragement throughout my stay at Hughes, and were always concerned with my well-being. I couldn't have asked for any better people to work for.

I would like to express my appreciation to Hughes for its commitment to the MIT Engineering Internship Program, and to education in general. The coordinators of the co-op program, including Mimi Shefrin, Marshall Thompson, and Gretchen Replogle, make every effort to make the students' stay here extremely enjoyable. Furthermore, Hughes managers like John Drebinger, Dick Remy, David Breuer, and Dick Pio, make it possible for a large number of students to benefit from and contribute to Hughes' technical expertise. It would have been impossible to complete this research without the resources, both in people and in facilities, that Hughes provided.

On the MIT side, John Martuccelli and Brenda McFadden run the Engineering Internship Program professionally, with the concerns of students in mind. They were always available to provide guidance, and to help solve administrative problems.

Charles Dolan, my immediate supervisor and the group head for Artificial Intelligence at Hughes Radar Systems Group, organized the scope of my research effort, made significant contributions to the design of the expert systems, and taught me a lot about AI. In addition, he provided sources for background information and valuable comments on thesis drafts, helping to make it a coherent dissertation. Though busy as a student himself and as manager of our group, Charlie always had time for discussions about my work or about AI in general.

Mark Young and Craig Lee, my co-workers and office-mates at Hughes, contributed greatly to the project. Mark designed and implemented two belief mechanisms used in the project, and worked with me on integrating them into the expert systems. He furthered my understanding of Bayesian and Shafer-Dempster probability theory, as well as made suggestions on the design of the expert systems. Mark undertook the unenviable task of reading drafts of my thesis, providing interesting comments and helping to eliminate excess verbiage.

Craig provided unparalled system support, amazingly answering any questions I had about our (reliable) Apollo machines, the LISP environment, or the VAX. Craig ported the ARF frame system onto the Apollos, made T-LISP compatible with FRANZ-LISP, and designed the frame system that we will use in future implementations of the object analysis expert system.

Both Mark and Craig are designing extensions to the expert system, including a sensor simulator that should make input to the system much easier.

I would like to thank a number of other people at Hughes, including Mike Harris, Jack Schwartz, Bill Hefenieder, Todd Marsh, Claudia O'Brien, Terry Bay, Lisa Jack, Dorene Grimmett, Mary Corridan, Tanya Greenwood, Mitzi Mori, and the others in the SAR Lab and APD who were always willing to lend a hand. The people are what make Hughes a truly enjoyable place to work.

I would like to thank Professor Antonio Elias, my thesis advisor at MIT, who sparked my interest in avionics applications of expert system and took the time to come to California to see my work first-hand. He also made a number of valuable suggestions about the content and organization of the thesis.

Finally, and most importantly, I would like to thank my parents for their guidance and love. They have always done what is best for me, and have set me in the right direction in life.

CHAPTER 1
AN OBJECT ANALYSIS EXPERT SYSTEM

Background

This thesis was completed at the Radar Systems Group (RSG) of Hughes Aircraft Company, El Segundo, California, in partial fulfillment of the requirements for M.I.T.'s Engineering Internship Program. EIP, as it is called, requires that a student spend a seven-month period at the sponsoring company (Hughes), completing master's degree thesis research on a project of mutual interest to both M.I.T. and the company. The project I chose originated out of my interest in systems engineering and artificial intelligence, my advisor's interest in expert system applications in the avionics environment, and Hughes' interest in both areas.

Introduction

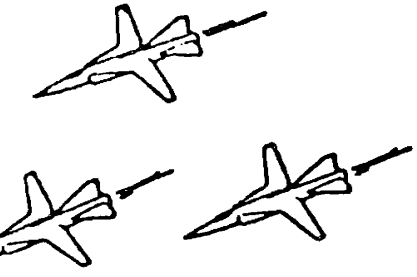
This thesis describes the development of a prototype expert system to perform airborne object analysis, the task of using returns from an airplane's sensors to infer type identification and associated tactical information about surrounding airborne objects. Using a set of if-then rules developed through sensor and aircraft research, the prototype system accepts a file containing a sequence of simulated sensor data and attempts to characterize approaching aircraft as completely as possible. Since sensor data becomes more plentiful and more accurate as objects approach, the identification algorithms provide better

ID over time. For example, at a range of 80 nautical miles (nmi), the expert system may be able to tell us only that a plane is high-velocity and powered by jet engines; at 45 nmi, however, when the plane is within the detection range of a greater variety of sensors, we may be able to deduce, for instance, that it is a MiG-class fighter, tracking two possible targets on its radar.

The figures on the next two pages characterize the expert system's classification ability. **Figure 1a** shows a typical scenario a fighter pilot might face, along with sensor returns available to him at a range of approximately 80 nmi. While the sensors provide a great deal of data on the enemy airborne objects, they provide no easily useable information *about* them. Just by looking at the sensor displays, the pilot has little idea about the identity, capability, or mission of the opposing planes, and he is too far away to visually assign them into formations. The pilot must therefore analyze the sensor returns himself, and from experience on past missions determine the characteristics of his opposition. This is time-consuming and diverts the pilot's attention. An alternative is to relegate the task to an object analysis expert system, which produces the results of **Figure 1b**.

From sensor data and its own knowledge base, the expert system has determined the identities of the planes, grouped

FIGURE 1A: CLASSIFICATION ABILITY WITHOUT OAS



INFORMATION PRODUCED BY SENSORS

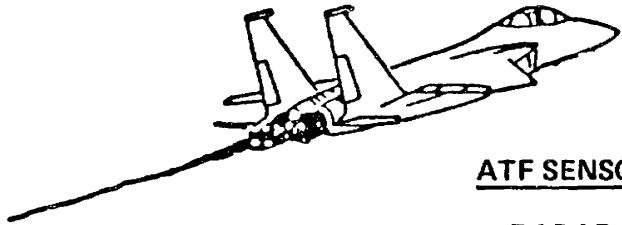
6 INCOMING OBJECTS

VELOCITY: *MACH 1.3*

IR TEMP: *1500 DEG, TOP GROUP*
700 DEG, LOWER GROUP

ALTITUDE: *30,000 FT, TOP GROUP*
15,000 FT, BOTTOM GROUP

RECEIVING RADAR FROM TOP GROUP



ATF PILOT MUST MAKE HIS OWN INFERENCES

ATF SENSORS

RADAR
INFRARED
RWR
FLIR

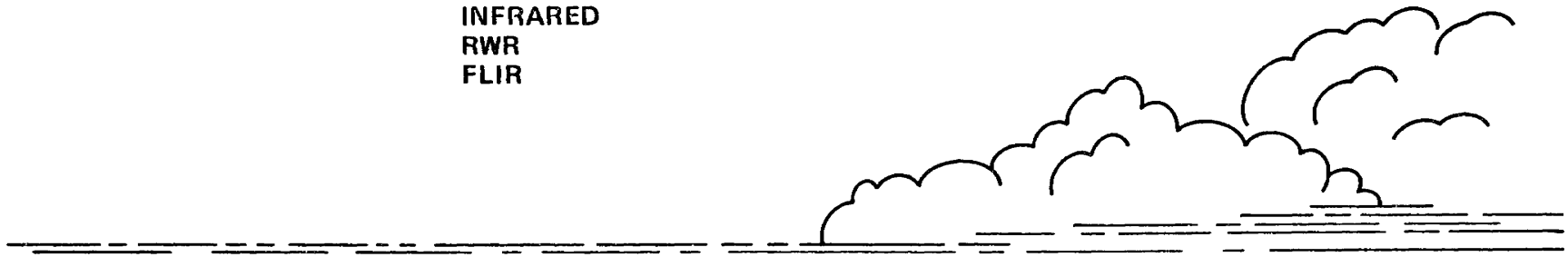
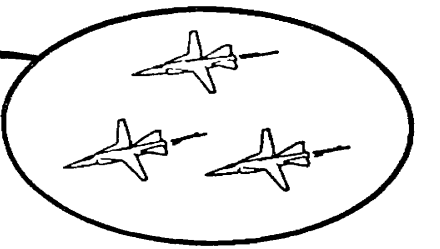


FIGURE 1B: CLASSIFICATION ABILITY WITH OAS

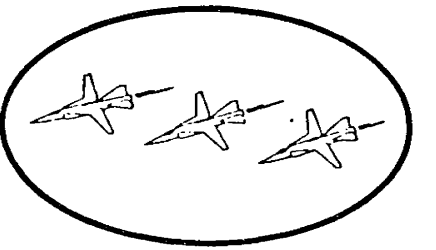
USING THE SAME SENSOR INFORMATION AS BEFORE, OAS INFERS:



NAME: FORMATION 1

CONSTITUENTS: *MIG-29 FIGHTERS*

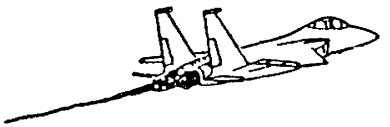
MISSION: *ESCORT, AIR-SUPERIORITY*
PHASE: *DASH*
POSS TARGET: *ATF*
EST. TIME ARRIVAL: *7 MINUTES*
TRACK: *ACCELERATING, CLIMBING*
EXPECT: *INITIATION OF CLIMBING ATTACK*
CAPABILITY: *LOOK-DOWN/SHOOT-DOWN RADAR ACCELERATES BETTER, TURNS WORSE THAN ATF*



NAME: FORMATION 2

CONSTITUENTS: *TU-95 BOMBERS*

MISSION: *BOMBING-RUN*
PHASE: *DASH*
POSS. TARGET: *BASE B, HEADQUARTERS*
EST. TIME ARRIVAL: *8 MINUTES*
EXPECT: *"POP-UP" TO DELIVER ARMAMENTS*
CAPABILITY: *LIMITED SELF-DEFENSE*



OAS MAKES INFERENCES FOR ATF PILOT

them into formations, determined possible targets, and hypothesized their mission and roles.[1] The expert system relieves the pilot of performing object analysis himself, while giving him useful information about the opposing strike force. (For more sample scenarios and actual runs of the system, see **Appendices A and B**.)

Scope of This Work

The development effort reported here entailed the design and implementation of two versions of the expert system, one rule-based [Shortliffe, 1976] and one demon-based [Rieger and Small, 1979]. Both use a **frame-slot-filler** structure [Bobrow and Winograd, 1977] to represent knowledge. The development of two systems was not originally planned, but as is often the case in expert system construction, the development cycle was extremely iterative. After construction of the rule-based system, which served primarily as a learning and feasability tool, we went to a demon-oriented system controlled by **procedural attachments** [Bobrow and Winograd, 1977], resulting in greatly improved efficiency and flexibility. Though the demon-based system (hereafter OAS2) is much more suited to the object analysis task, I have not omitted discussion of our first effort (OAS1) in this thesis. We learned a great deal

---

1. See the **Rule Development** section of **Chapter 3** for a discussion of the criteria for these inferences.

about the knowledge domain, belief mechanisms, and knowledge structure by doing the implementation, and OAS1 served as an important bridge to the design of OAS2. As predicted in [Hayes-Roth, Lenat and Waterman, 1983], it is usually the case that after finishing the first cut at an expert system, it is best to throw it away. We felt it was important not to be confined to design limitations and decisions made when less knowledge was available about the problem domain and system-building tools.

Prior to the design and implementation of the expert system, preliminary tasks of knowledge engineering and rule development had to be performed. This involved research on pilot decision rules, sensor capabilities, and aircraft characteristics. One problem we faced is that there is no human expert currently performing the object analysis task. In most development efforts, a human expert is available to participate in the knowledge engineering, and to evaluate the system as progress is made. However, no human currently integrates all the sensor returns in a tactical fighter. A possible exception to this is the naval flight officer in an F-14, but even then the level of sensor information fusion is not as great as that required for an object analysis system.

A Guide For The Reader

The remainder of this thesis is organized in a manner that

reflects the development path followed during the construction of OAS1 and OAS2. Certain tasks were necessarily performed prior to others, and it may be beneficial to the reader to share the insight gained as research progressed to design, then implementation, and finally evaluation. Precisely because expert system development is so application-dependent, it is important to discover how the methodology of design affected development.

Chapter 2 provides background material on object analysis and expert systems. Requirements of object analysis and justification for its need are discussed. A brief history of expert systems is presented, along with requirements for their successful construction. Interaction between object analysis and expert systems is also discussed. The section concludes with an examination of the uses of artificial intelligence (AI) in the avionics environment, and the advantages of using expert systems to solve the object analysis problem.

Chapter 3 discusses the knowledge engineering aspect of expert system development, including research on sensor capabilities and airborne-object characteristics. The rule-development process is discussed, and a list of the rules used by OAS1 and OAS2 is presented.

Chapters 4 and **5** discuss the design and implementation of

OAS1 and OAS2, respectively. The design of each system is presented in detail, along with an evaluation of the results. The impact of OAS1's shortcomings on the design of OAS2 is discussed.

Chapter 6 presents a summary of results, and a review of the main discoveries of the research. This section also indicates areas where future research may be oriented, points out the need for expansion of the rule set, and enumerates requirements and justification for an explanation facility. Possible extensions to the user interface and expansion of the scenario and airborne-object domain are discussed.

CHAPTER 2
OBJECT ANALYSIS AND EXPERT SYSTEMS: SOME BACKGROUND

Object Analysis

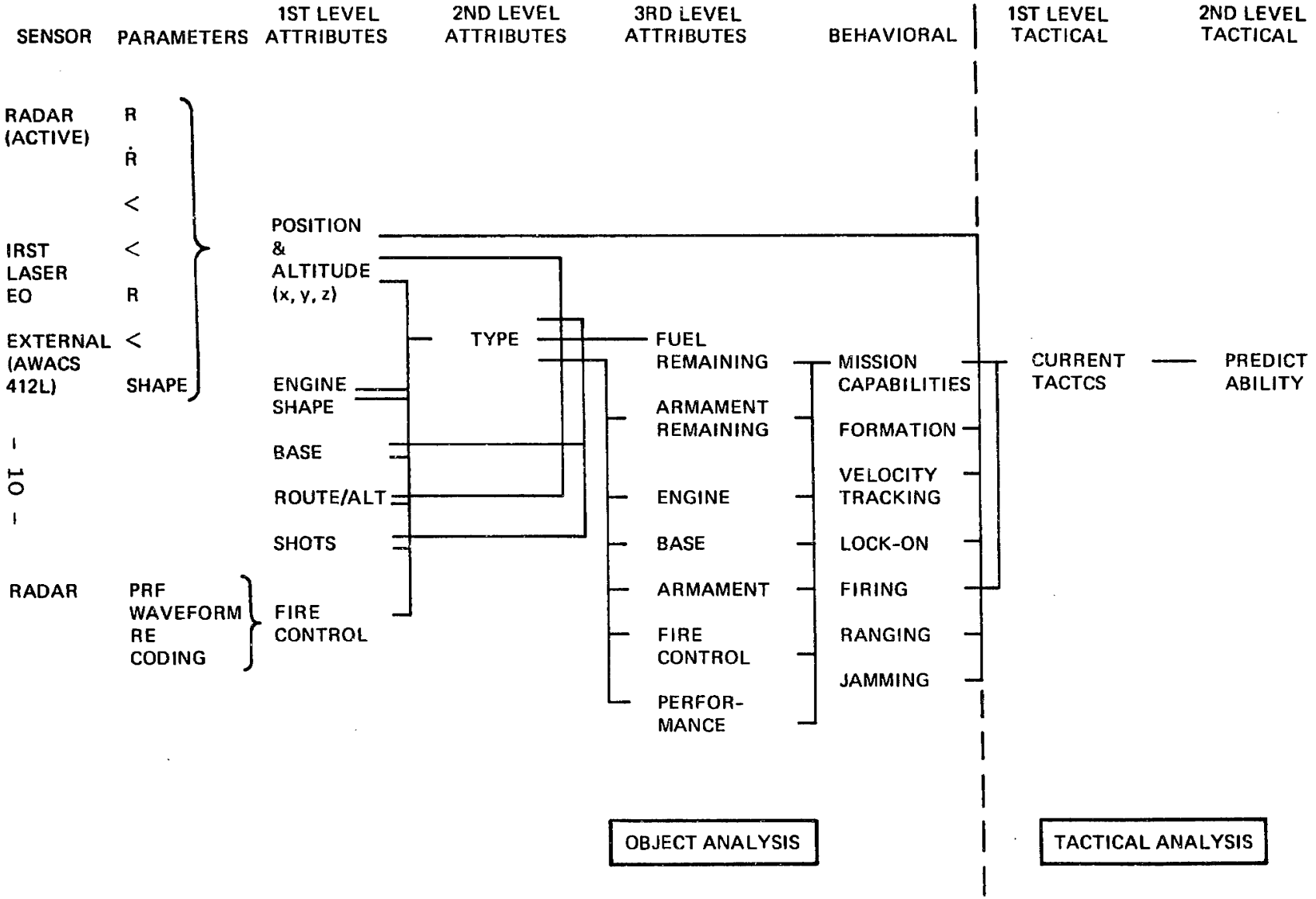
The purpose of object analysis is to combine low-level sensor returns into useful, higher-level information. As shown in the feature hierarchy table of **Figure 2**, sensors provide, among other things, information on a surrounding object's range, velocity, shape, and radio frequency (RF) emissions. Upon receiving enough information about a single target's "track",[2] **first level attributes**, such as engine type, possible point-of-origin (base), and mission route can be inferred.[3] When, in turn, we have acquired enough first level attributes on a track, we may be able to infer type or class of craft, and thus capability (fuel, armaments, fire control, performance). **Behavioral attributes** can be better deduced by examining groups of planes or objects, rather than just one at a time. For instance, the formation and maneuvers of a group of planes may indicate their types, roles, and possible missions. As one example, if fighter planes are escorting bombers, the former must execute "S" turns periodically to ensure that they do not overtake their slower counterparts. This type of motion can be detected by radar and passed to the object analysis system,

---

2. "Track" is the military term used to indicate an airborne object's flight path through the air.

3. Inference rules are discussed in the **Rule Development** section of **Chapter 3**.

FIGURE 2: FEATURE HIERARCHY TABLE



which may be able to hypothesize that the incoming group of planes is a ground attack squadron of bombers escorted by fighters. With knowledge of the ratio of bombers to fighters and spacing in a typical enemy formation, it may be possible to estimate the size of the incoming raid and allocate tactical resources accordingly. Decisions on maneuvers, strategy, or resource allocation, however, have been left for the tactical situation assessment task, to be developed in the future.

In current avionics systems, object analysis as described above is done implicitly by the pilot as he flies his plane. There are separate displays inside the cockpit for each sensor, and the pilot glances at them periodically, making interpretations based on experience. There is not, however, a computer or human dedicated to the object analysis task. The reason for the low priority of object analysis is twofold: (1) the current range of air-to-air missiles is not as great as the detection ranges of sensors, so even if a plane could be identified at long range, it could not be fired upon; and (2) current rules of engagement require visual identification of the enemy before attack. This thesis research, however, is in anticipation of the time when advances in missile technology (and sensor research) will permit air battles to be fought beyond visual range (BVR). Under BVR conditions, quick and accurate assimilation of the vast amounts of sensor data that will be available will provide a distinct advantage.

Expert Systems

Expert systems have been the most successfully applied AI technique to date, resulting in operational systems that solve symbolic mathematical problems, diagnose disease, and perform geological prospecting. (See [Hayes-Roth, Lenat, Waterman, 1983], [Gevarter, 1983], and [Clancey, 1984] for survey discussions of expert system applications; the section below entitled **Related Expert System Research** examines other work oriented towards military applications.)

Simply put, an expert system is a computer program that takes as input known or partially known information, examines a knowledge base for rules that use this input, and deduces new information that hopefully helps to solve the problem. Required steps for successful expert system construction include knowledge engineering (learning about the problem and the "knowledge" that constitutes expertise in the field), rule development (combining results from knowledge engineering with protocols of human experts to form rules a computer can interpret), inference engine design (developing a way to use the rules formulated), and knowledge structure (designing an efficient way to represent the knowledge used in solving the problem).

Often the most difficult part of expert system development is the embodiment of knowledge in the system. A doctor trained

in the diagnosis of infectious diseases, for example, may not always be able to explain his exact methodology. Many of his techniques may have evolved over years of practice on hundreds of cases, and may be based on intuition or rules of thumb. Even when one is able to extract a protocol from an expert, there is often difficulty in translating his manner of expression and terminology into accurate representations in the expert system knowledge base. The expert system-developer, who is usually relatively inexperienced in the problem domain, thus faces the problem of correctly translating information from an expert who may know little or nothing about the requirements of computers.

Related to translation is the **conflict resolution**[4] problem. Here the experts decision making strategies must be translated into probabilistic algorithms, requiring the use of confidence factors in evidence and rules. For example, a doctor may say, "If a patient's throat culture is positive on Test A, it usually indicates that the patient has strep-throat" [Shortliffe, 1976]. The doctor has expressed his "degree of belief" in this rule by the word "usually". For use in an expert system, however, this English-oriented confidence measure must be translated into a number. Furthermore,

---

4. Conflict resolution, as the name implies, requires determining the "best" solution when evidence indicates different answers.

translation conventions vary with the problem domain, so the knowledge engineer must become thoroughly familiar with the expert's terminology.

Expert System Belief Maintenance

Once we have acquired confidence factors, we must determine how we are going to combine them. There are two related problems to solve: **confidence propagation** and **assertion combination**. Using the strep-throat rule of the previous paragraph as an example, confidence propagation is the task of determining how confidence factors in the evidence ("Test A is positive") and the rule itself combine. For instance, if Prob(Test A is positive) = 0.6 and the "belief" in the rule is 0.7, then we must determine the confidence that "patient has strep-throat". One popular confidence propagation method simply multiplies the two factors, resulting in a confidence of 0.42. This simple method, however, is not sufficient when there are multiple sources of evidence and multiple conclusions [Lowrance and Garvey, 1983, p.14]. (See the **Belief Maintenance** section of **Chapter 4** for more details.)

The second problem is the combination of assertions. If the rule above asserts "patient has strep throat" with confidence 0.42, and a different rule asserts "patient has tonsilitis" with confidence 0.75, there is a question of how to combine these assertions and determine the patient's disease.

The solution has been dealt with in a number of ways, including ad-hoc [Shortliffe, 1976] and classification methods [Ben-Basset, 1980] [Clancey, 1983], Bayesian inferencing [Duda, Hart, and Nilsson, 1976], and Shafer-Dempster theory [Garvey and Lowrance, 1983] [Dillard, 1983].

Some assumptions of a belief-maintenance mechanism, it should be noted, can be blatently wrong. For example, a standard assumption about two competing hypotheses about which no evidence has been received is that they are both equally likely (e.g., we are trying to decide if its sunny or raining, but we can't look out the window (no input), so we assign each proposition probability 0.5). In a real-world situation, however, when a human expert is weighing two alternatives, he does not usually assume equal likelihood [Garvey, Lowrance, and Fischler, 1981, p.320]. This is because he may have prior knowledge that is unavailable to the belief mechanism (e.g., it is summer in Los Angeles, so it's probably not raining). These are issues that make accurate belief-maintenance an especially difficult problem for expert systems. The methods we have used for our implementations are discussed in **Chapters 4 and 5.**

Knowledge Representation and Reasoning

The software tasks of designing an effective knowledge structure and inference engine have their own set of

requirements. The structure must be both complete enough to hold all the data necessary to solve the problem, and flexible enough to allow extensions to the knowledge base. In addition, the most-used data should be easily accessible. One common method of holding knowledge, used in both of the expert systems of this thesis, is a frame system [Bobrow and Winograd, 1977] [Minsky, 1975] [Edwards, 1984]. Frames hold information about classes of objects in **slots**, and provide **accessors** to put and get **values** from slots. To hold information about airborne objects, we could define a frame pattern by:

```
(define-frame airborne-object
     (slot track#)
     (slot xyz-location)
     (slot velocity
     (slot #engines)
     (slot ID)
     (slot parent-formation)
```

Then when we learned about a plane, we could:

```
(set PLANE1
     (make-instance airborne-object
          (track#              1                )
          (xyz-location        (+35 -2 10000))
          (velocity            Mach-1.2         )
          (#engines            2                )
          (ID                  fighter          )
          (parent-formation    Attack-Group-A)))
```

This piece of code **instantiates**[5] an object of type airborne-object, and enables us to access the values contained

---

5. Instantiate means to create an instance (copy) of a frame pattern.

in PLANE1, or put in new slots or values. In addition to providing a structured way to store information, frames allow hierarchical organization in a parent-child structure. If, for example, we found ourselves detecting a large number of fighter airborne objects, we could define a new frame subordinate to the airborne-object frame called **fighter**, which would hold airborne object data specific to fighter craft, while maintaining properties all airborne-objects had in common. Then it would be easy to group, say, all airborne objects in the same formation with velocity greater than Mach 1.0, whose ID is also fighter. Frames allow design of knowledge structures that represent real-world ordering of information.

The design of an expert system's rules and inference engine influence its extensibility, efficiency, and ability to reason (i.e., deduce new information from what it already knows). To aid system evaluation, rules should use the terminology of human experts, and be designed to embody a single concept of the problem-solving method, avoiding, if possible, complicated multi-part rules. Furthermore additions to the rule base should be easy to make. An expert system's inference engine is the piece of code which cycles through the rule base searching for the implications of new evidence, given the current knowledge state. It can be as simple as a forward-chaining engine that checks the "if" part of a rule, and if its true, asserts the "then" part of the rule. Alternately, it can be a

backward-chaining engine that starts with a goal and generates evidence that satisfies the goal. Inference engines with **backtracking** can retract assertions that are later found to be incorrect. An ideal inference engine should not have to be modified as the rule base is extended, should allow generalized antecedants and consequents (if- and then-parts), rule base indexing, and efficiency checks to ensure that rules that are not affected by a particular piece of new evidence are never even examined. The degree to which these various criteria are satisfied by OAS1 and OAS2 is discussed in the **Design** sections of **Chapters 4** and **5**.

Related Expert System Research

There have been a number of research efforts in the military applications area aimed at the development of automated methods for performing tactical situation assessment [Spain, 1983] [Geschke, Bullock, and Widmaier, 1983], sensor combination [Waltz, 1981], and report assimilation [Lenat, Clarkson, and Kiremidjian, 1983]. None, however, are directed at object analysis as a precursor to tactical situation assessment. In the airborne environment, at least, identification of the opposition greatly influences tactical decisions. This will become even more true with the onset of BVR encounters. Most previous research efforts have either focused limited attention on object identities, or presumed they are given in the system.

In addition, previous efforts have dealt with the ground-based or naval environment, rather than the airborne environment. Airborne object analysis presents a host of problems not encountered in other areas, such as the possibility of rapid maneuvers, the sophistication and quantity of sensor reports, the degree to which jamming and counter-measures are utilized, and the need for very fast decision-making.

Furthermore, since extensions to the object analysis task plan to integrate diverse non-airborne information sources, such as ground intelligence information and JTIDS[6] data, much of the previous work may contribute to the system we have developed.

Good problems for Expert System Application

In examining a candidate problem for expert system application, one requirement is that the solving methodolgies be well-known. An expert should be able to identify a plan or sequence of steps that he follows in solving the problem. In addition, the problem should not be trivial for an expert to solve (else why build an expert system to do it), and it should not require tremendous effort (otherwise it will be difficult

---

6. JTIDS: Joint Tactical Information Display System, an armed forces network of integrated intelligence information.

to embody the complexity in the knowledge base). The next section discusses the specific application of expert systems to object analysis.

Why Use AI Techniques For Object Analysis?

Object analysis is the first step towards the future goal of automating many of the tasks that pilots must now perform themselves. As discussed above, this will become increasingly important with the onset of BVR air-encounters and airspace-saturation. The allocation of low-level tasks to intelligent subsystems illustrates the **pilot-as-system-manager** concept, the objective of which is to reduce the pilot's workload by presenting him only with information relevant to the mission at hand. For example, if enough fuel remains to complete a mission, there is no need to inform the pilot of this fact unless he specifically requests it. By distributing low-level tasks to "electronic co-pilots", the pilot is left free to fly his plane, communicate with friendly forces, and make tactical decisions. For this structure to work, however, the pilot must trust the decisions of the co-pilots and be able to easily communicate with them. This explains the desire to implement the co-pilots with artificial intelligence techniques, using natural language processing and speech synthesis for communication, and expert systems for their reasoning and explanation ability.

There are several characteristics of expert systems that make them particularly attractive for application to the object analysis task. First, the explanation facility that exists in most operational expert systems allows the user to trace the inference path of the expert system. The pilot could query the system during use, determining what evidence has been received, what rules have fired, and what knowledge has been inferred. Over time, the pilot would develop trust in the results the expert system provides, and he would be able to "fine-tune" the system to his own particular needs.

Another desirable characteristic of expert systems is their ability to provide requests for needed evidence. For example, when the inference engine is checking the rule base, it can identify those unavailable pieces of information which would allow a rule to fire if they were present. This is an ideal input to an avionics sensor manager, which determines how to allocate sensor resources upon the many surrounding airborne objects. An object analysis expert system could report to the sensor manager, "If I had the radar cross-section of target 22, I could identify its class." This request provides a valuable input to the sensor manager, whose job is to prioritize the acquisition of sensor data. Given constraints on time-on-target and the information that sensors can provide, the sensor manager attempts to maximize the utility of sensors [Blackman and Broida, 1983].

In addition to being able to deal with nonexistent information, expert systems have an advantage over non-AI-based computer systems in that they can deal with uncertain information in a manner rivaling that of humans. Some sensors, for instance, may be only intermittently operational due to excessive range, poor weather conditions, or jamming by the opposition. Furthermore, a sensor may not return 100% accurate data due to operating conditions or limited time-on-target. In either case, it is desirable to have a system that still provides us with results, despite the partial lack of information. Though not able to specifically identify a plane as, say, a Tu-95 bomber, an expert system may at least be able to tell us that the object is a relatively slow plane with four turbo-prop engines. The pilot thus has a decision aid even if complete information is not available.

Summary

Object analysis is a problem that is well-suited to expert system application. In one sense it is difficult because there are no human operators explicitly dedicated to the task--pilots do it intuitively and incompletely. However, object analysis is essentially a classification problem of grouping planes based on evidence, and there is a wealth of information available on aircraft characteristics, sensor abilities, and

enemy tactics. With the advent of sensor fusion[7] and the inevitable advances in sensor technology, more and more information will become available, and computer systems with intelligence will be needed to sift through all of it. The next chapter examines some of the knowledge engineering accomplished in order to demonstrate the applicability of expert systems to the object analysis task.

---

7. Sensor fusion involves control and combination of multiple sensors for tracking and identification functions. See [Waltz, 1981]

CHAPTER 3
KNOWLEDGE ENGINEERING AND RULE DEVELOPMENT

Introduction

Though expert system development is highly application-dependent, at least two tasks must be completed for their successful construction. The first is knowledge engineering: identifying the structures appropriate to store the specific knowledge required in an application. (This topic is discussed in the Block Diagram sections of **Chapters 4** and **5**.) The second task is transfer of expertise, with the associated task of knowledge-gathering. For object analysis, this involves interviewing pilots and operations analysts about air-to-air situation assessment, and embodying their decision methodology in if-then rules. In addition, knowledge-engineering requires researching aircraft characteristics, sensor capabilities, and plausible scenarios, for the purpose of categorizing them and developing a suitable knowledge representation scheme.

The knowledge engineering and rule development tasks are highly interrelated: as new rules are proposed, relevant data must be entered into the knowledge base; similarly, researching knowledge that characterizes the object analysis domain enables us to examine new areas for possible identification rules.

In the object analysis task we are trying to characterize an unknown airborne object using sensor data. Therefore, we must deal with two questions:

1. What characterizes an object as being an aircraft of type A, and not of type B (or C or D)?
2. What types of sensor information are available to identify objects?

The next two sections discuss answers to these questions.

Aircraft Research

To develop a knowledge base, we investigated the types of airborne objects we would be identifying in a battle environment. Our sources were primarily public-domain, and included *Jane's All The World's Aircraft* [Taylor, 1985], *Aviation Week and Space Technology* articles [1-9],[8] and other defense articles [10-12]. In addition, we checked classified sources of information to ensure that the data we would be using in our demonstration system would not differ significantly from that of an operational expert system. Of course, a future object expert analysis system will be able to take advantage of classified sensor sources to perform more powerful inferencing. In this thesis, however, we are primarily concerned with the system's structure and problem-solving methodology.

---

8. See **References.**

Of the dozens of possible types of airborne objects in a typical battle scenario, we chose to limit ourselves to the four classes listed below for a demonstration system. We have ignored, for example, helicopters, surveillance and reconnaissance planes, surface-launched missiles, and decoy objects. These objects can be added rather easily to the expert system, once the proper research has been done, and should greatly enhance the utility of the system. We have also not done in-depth research on each of the classes of objects listed below. There are probably a hundred different kinds of fighters alone, each with its own radar, fire control system, and capability. Below, for each class of craft are listed some characteristic features, usual function and capability, and some specific IDs that make up the class.

Fighter

Fighters are high speed, versatile craft that are usually equipped with turbojet engines. They can fly air-to-air, air-to-ground, or escort missions, carrying six to twelve air-to-air rockets or bombs. Characteristics of these planes include: high velocity, engines with afterburner capability, medium radar cross-section, small wingspan and length, very good maneuverability, high-altitude capability, and multiple radar capability. Specific planes include American F-14s, F-16s, and F-18s, and Soviet MiG-21s, MiG-25s, Su-27s, and MiG-29s. Each specific plane can be specially outfitted to perform special missions.

Bomber

Bombers, in general, are low or medium-altitude, average-velocity ground-attack planes. They have limited self-defense capability, and are usually escorted by fighters. Additional characteristics include: more engines that other planes (up to eight), extremely large combat radius, low-flying or terrain-hugging flight path, jet engines, large wingspan and length, and ground-mapping radar. The B-52 and B1 are U.S. bombers; Tu-22 and Tu-95 are Soviet bombers.

Rocket

Rockets (or missiles) can be air or surface-launched, and include everything from anti-tank to ballistic cruise. For now, we are concerned with those that would threaten ownship.[9] These include air-to-air or anti-aircraft surface-launched rockets. Rockets are generally small and extremely fast. They may contain their own radar for guidance, or may require "illumination" by the launcher's radar in order to hit a target. Surface-launched rockets are usually larger, have lower initial speed, greater range capability, and begin their flight from zero elevation (the ground). Air-to-air missiles are smaller and shorter-range, and have an initial velocity equal to that of the jet that launched them. A U.S. rocket is the Sidewinder. Soviet rockets include the AA-7 and AA-9.

---

9. Ownship is a military term meaning "our own airplane".

Transport

Air-transports carry troops, supplies, and equipment from supply posts to the front lines. They have limited self-defense capability and are usually escorted by other aircraft. Two main types exist: subsonic, which have propeller engines and are relatively slow, and supersonic, which have jet engines and are comparatively fast. Both types have large radar cross-sections and wingspans, limited radar capability, and medium-altitude capability.

Sensor Research

In order to construct realistic sequences of sensor returns (trackfiles) to drive the scenarios, we investigated what sensors would typically be available on a fighter, what information they could provide, and what their capabilities were (detection range, resolution, and confidence as a function of range). Sources for our results included sensor and avionics experts at Hughes [13,14], books and periodicals on sensor technology [Stimson, 1983], and internal Hughes reports [Blackman and Broida, 1983] [Goltzman, 1984]. Since performance data is classified, we are not using exact detection ranges or full sensor capability. For our purposes, however, accurate "numbers" are not important. We need only a representative idea about what kinds of information we can get from sensors, and in what sequence it becomes available. The following paragraphs describe the sensors that would probably

be available on a fighter such as the ATF.[10] We have speculated about some sensor's future capability, since the expert system will not be in operational use for several years.

Radar

Radar is a standard, long-range sensor available on nearly every combat aircraft. It provides range, range-rate (velocity), and bearing data[11] on surrounding objects. In addition, it provides radar cross-section (RCS), an indication of the size of an airborne object. Research is underway to provide radar with even more advanced capabilities. A radar's useful range can be in excess of 100 nmi, and its angular resolution and accuracy can be quite good. Disadvantages of radar include its susceptibility to enemy jamming and electronic counter-measures (ECM), its performance degradation by clouds and inclement weather, and the fact that it is an active (emitting) sensor.

---

10. ATF stands for Advanced Tactical Fighter, the Air Force's designation for its next generation of fighter aircraft, in which AI technology and electronic co-pilots will be used. Background material on ATF is provided in [Kinnucan, 1983]. In this thesis, ATF is also referred to as ownship.

11. Bearing (or heading) as used in this thesis is the angular position relative to ownship.

IRST (Infrared Search Track)

An IRST obtains the heat characteristics of objects by measuring the frequency and intensity of their infrared radiation. This information indicates the temperature and size of the engine, whether its accelerating or not (by keeping a time history of temperature), and what mode its engine is in (afterburn, for example). The angular resolution of IRSTs is particularly good, and depending on the wavelength used, detection ranges can extend to 50 nmi. IRSTs can have problems when looking down upon an object, since the ground appears as background clutter, but since they are passive sensors, they are impossible to detect and are relatively immune to direct jamming.[12]

FLIR (Forward-Looking Infrared)

FLIR is a TV-like device that can provide visual data at night. It gives temperature signatures like an IRST, in addition to information on the shape, length, and width of objects. Shape information might include special features of an aircraft, such as its silhouette, external fuel pods, or weapons stores. FLIR is a relatively short range sensor (20 nmi) and is practically immune to jamming.

RWR (Radar Warning Receiver)

RWRs provide information about the electromagnetic emissions of enemy objects. This includes data on radar and fire control systems, such as modulation frequency,

---

12. Flares, however, can be used to distract infrared sensors.

radio frequency (RF) band, pulse repetition frequency (PRF), waveform, pulse type, and mode. In addition to helping characterize a particular type of plane, this data may indicate where the enemy's radar is searching or tracking, and whether he is preparing to attack. RWRs are susceptible to jamming and frequency agility techniques.[13] Useful range is greater than 150 nmi.

There are a great many sources of intelligence information other than that provided directly by onboard sensors. AWACS (Airborne Warning And Control System) planes carry very sophisticated electronic equipment and can provide more precise information at greater detection ranges than ordinary fighters. JTIDS, currently under development, will coordinate communications and sensor data from ground, naval, airborne forces. Other planes in the same formation as ownship can provide evidence that supports ownship's sensor data, or pinpoints and object's position.[14] Ground-based friendly forces can detect troop deployment, supply line movement, and launchings at enemy airbases. Finally, pre-flight briefings can give weather conditions and positioning of enemy jamming sources (both of which degrade sensor performance), as well as up-to-date deployment information (number and types of planes)

---

13. Frequency agility is the tactic of rapidly changing the radar's base frequency to prevent opposing radar trackers from "locking on" to it.

14. Two planes correlating their respective bearing data can more easily pinpoint the location of an enemy object.

at enemy bases. While some of these additional information sources are implemented in the latest version of the expert system (OAS2), much has been left for future development. The classification ability of the expert system should greatly increase with the addition of these sources.

When a polished object analysis module is present in the future, it is hoped that results from running the expert system on real scenarios will provide valuable feedback to those investigating sensor technology. By analyzing the results of the expert system, we may be able to determine those sensors with the highest utility, and help to shape the sensor suite of advanced fighters. In addition, we might identify what additional sensor data would be most useful in identifying planes, thereby helping to direct future sensor research.

Scenario Research

A third knowledge engineering area we researched is the operational and flight profiles of enemy planes, including missions, flight patterns, radar techniques, and mission phases. This area contains the most unpredictable and least reliable, yet potentially most valuable information available. We investigated only a small amount of the available information, as much of it is classified. A summary of the information that influenced rule development is given below.

Planes typically fly in groups, and can be judged "in formation" by physical proximity, or by similarity in maneuvers or predicted targets. There are a tremendous variety of possible missions for a formation, including **bombing-run**, **air-superiority**, and **defense suppression**. A formation with mission bombing-run would include bombers to deliver armaments to ground-based targets, plus escort fighters to provide self-defense capability. Air-superiority formations consisting of fighters might have a mission to eliminate opposing fighters protecting an airbase or a formation of ships. A defense suppression mission could include jets with air-to-surface missiles whose task would be to cripple opposing ground forces in preparation for an air or ground invasion. With identification of enemy mission comes clarification on the number of planes in the formation, what friendly resources will be needed to match the attack, possible targets (based on heading), and predicted future tactics.

Each possible mission above has different "phases", with characteristic maneuvers and tactics. A typical sequence of phases in a mission is: launch, climb, cruise, dash (to target), attack, retreat, cruise, land. With identification of enemy mission phase comes knowledge of: ETA (estimated time of arrival), future tactics, maneuvers, and velocities. More aspects of scenario research will become evident in the next section, **Rule Development**.

Rule Development

In translating the results of knowledge engineering into usable rules, we examined each measurable a sensor could provide and determined what intermediate or direct effects it could have on the identity of an airborne object. Some rules use only a single sensor parameter in the antecedent (if-part), while others have antecedants that use two or more sensors. Infrared emissions, for example, can somewhat characterize a plane by itself, since the heat-content of an airborne object indicates the type of engine it is using. Velocity alone, on the other hand, does not provide an indication of identity. However, when it is coupled with a second sensor parameter like RCS, we can draw conclusions about ID.

The consequents (then-parts) of rules either assert ID information directly, as in the velocity-RCS rule above, or they "post" intermediate results which may later trigger other rules. This is the case with the IR temperature rule, which deduces information about engine type. Later, another rule-firing will relate engine type to identity.

The rules listed below have been divided into categories based on the sensor returns they use and the effect they have when they fire. Where needed, a short explanation or justification follows a rule. Keep in mind that rule-confidence values have been omitted from this list, but in actual use, assertions like "ID is bomber" become "ID is

bomber, confidence = n", where $0 \leq n \leq 1$. (A consequent's confidence value is a function of the "belief factor" assigned to the rule, and the probability with which the rule's antecedent is true.) Furthermore, any numbers used in a rule are merely representative of a threshold whose exact value can be determined by experts when they evaluate the system. At the end of the list are a few rules provided by Defense Systems Corporation (DSC), a company under subcontract to Hughes to perform expert interviewing. DSC presented pilots with simulations of real scenarios and collected protocols from them.

Not all the rules listed here ended up being implemented in the first expert system because of design constraints, and in the second expert system because of time limitations. All rules here can be implemented in the second system, however, and should greatly increase its power.

Physical Property Rules

This section contains rules that deal with the physical attributes and capabilities of airborne objects, such as velocity, dimensions, climb-rate, and radar cross-section.

```
IF velocity < Mach 0.8
     AND radar cross-section is large
THEN object ID is bomber

IF velocity < Mach 1.5
THEN object ID is not a rocket
```

```
IF velocity > Mach 2.0
     AND radar cross-section is small
THEN object ID is rocket

IF velocity > Mach 1.5
     AND radar cross-section is medium
THEN object ID is fighter
```

These four rules indicate that different classes of planes have different maximum velocities. When combined with radar cross-section, which gives an idea of the size of an aircraft, we have rules that say, "Fast and very small objects are probably rockets" and "Medium-size, slow objects are probably bombers". Further rules like these could be based on our knowledge of the typical velocities of aircraft in different phases of a mission.

```
IF |climb-rate| > 5000 ft/frame
THEN object ID is fighter

IF delta-velocity > Mach 0.4/frame
THEN object ID is fighter
```

These two rules say the the climb-rate and acceleration-rate of fighters is greater than that of other aircraft.

```
IF IR temperature < 500 deg
THEN engine type/mode is jet

IF IR temperature < 1100 deg
THEN engine type/mode is turbojet

IF IR temperature < 2200 deg
THEN engine type/mode is afterburn

IF IR temperature > 2200 deg
THEN engine type/mode is rocket

IF delta-IR temperature > 200 deg/frame
THEN object is accelerating
```

IF delta-IR temperature < -200 deg/frame
THEN object is decelerating

IF |delta-IR temperature| < 200 deg/frame
THEN object is holding speed

This set of rules demonstrates how infrared temperature returns, which give the "heat content" of an object, indicate engine type, since engines are a plane's main source of heat. In addition, keeping track of IR temperature changes over time shows whether an object is accelerating, decelerating, or holding speed [Goltzman, 1984].

IF number of engines[15] = 2
THEN object ID is fighter

IF number of engines = 4
THEN object ID is bomber

IF length[16] < 70 ft
THEN object ID is fighter

IF length > 70 ft
THEN object ID is bomber or transport

IF width < 10 ft
THEN object ID is rocket

IF width < 70 ft
THEN object ID is fighter

IF width > 70 ft
THEN object ID is bomber or transport

The above rules deal with the physical characteristics of

---

15. Number of engines is an observable not yet provided by any sensor, but future indications are that it may soon be available.

16. Length and width are provided by FLIR.

planes: bombers usually have four engines and a large wingspan; fighters are smaller and have two engines.

```
IF engine type/mode is rocket
THEN object ID is rocket

IF engine type/mode is turbojet
THEN object ID is fighter

IF engine type/mode is afterburn
THEN object ID is fighter
     AND object is accelerating
     AND mission phase is dash

IF engine type/mode is jet
THEN object ID is not a Tu-95 bomber
```

These rules associate engine type with ID and, in one case, mission phase.

Radar-Oriented Rules

This section contains rules that deal with properties that can be detected with ownship's radar, such as enemy radar type, PRF, modulation, frequency, and scan pattern. Shown here is just a small subset of the many types of rules that can be garnered from examining a radar's characteristics.

```
IF object is emitting
THEN object ID is probably fighter
     AND object ID is possible bomber
         or other aircraft using TF/TA[17]

IF radar altimeter emissions are detected
THEN aircraft class is reconnaissance
```

17. TF/TA: Terrain-Following/Terrain-Avoidance (i.e., map-of-earth operations).

Radar altimeters are used by a few kinds of aircraft to determine their own altitude [14]. Since this is an especially scarce occurrence, it makes a good rule for identification.

```
IF radar is Foxfire
THEN object ID is Su-27 fighter

IF radar is Hi-lark
THEN object ID is MiG-29 fighter
```

Foxfire and hi-lark are two types of radar known to be used by Su-27 and MiG-29 fighters [1]. They have special characteristics that can be detected by our own radar.

```
IF no radar is detected
  AND range is small
THEN object ID is bomber

IF aircraft is approaching
  AND PRF is "sawtooth"[18]
THEN radar mode is STT[19]

IF PRF is medium
  AND timing is burst
THEN radar type is SAR[20]

IF radar scan pattern is spiral
  OR raster
THEN radar mode is TWS[21]
IF radar scan pattern is conical
THEN radar mode is SST[22]
```

---

18. A sawtooth pulse repetition frequency resembles a ramping function, alternately rising and falling.

19. STT: Single Target Track

20. SAR: Synthetic Aperture Radar, a very high resolution ground-mapping radar.

21. TWS: Track-While-Scan

22. SST: Single-Scan Track

IF radar return is chirped[23]
THEN radar type is SAR

IF radar pattern is one fan and several slant beams
THEN radar mode is GCI[24]

Scan patterns refer to the path by which an enemy radar searches a given air-space. If enough readings are taken of the direction in which an enemy radar is "pointed", we can, over time, determine its scan pattern. From scan pattern we can determine radar mode, such as track-while-scan, which may provide clues to the object's intentions, depending on his mission phase [13] [14].

Target-Oriented Rules

Target rules deal with the criteria for airborne objects to be considered "in formation", along with the determination of targets, maneuvers, and missions of those formations. These rules border on tactical situation assessment, to the point of determining the intent of opposing planes. The next step, to be implemented by future electronic co-pilots, would be the determination of ownship's tactics from enemy intent.

IF dist(OBJ_X, OBJ_Y) < 5 nmi
    AND delta_altitude(OBJ_X, OBJ_Y) < 5000 ft
THEN OBJ_X and OBJ_Y are in the same formation

---

23. A pulsed method of sending radar signals.

24. GCI: Ground-Controlled Intercept

This rule indicates that two planes are in the same formation if they are within 5 nmi horizontally and 5000 feet vertically.

```
IF formation is heading towards a friendly base
THEN base is a target of the formation
```

The above rule is implemented by comparing the line-of-sight (LOS) between the formation center and a friendly site, with the heading of the formation. If the difference in angle is within a certain threshold, that site is considered a possible target of the formation.

```
IF a fighter is detected
THEN hypothesize escorted aircraft according to
     targets on heading
```

This rule indicates that fighters, in general, escort other aircraft, and the number escorted may be determined by the "firepower" necessary to attack a friendly base. In other words, if a fighter is detected in Formation A, and it is heading towards Friendly Base B, and on a normal bombing run it would take three bombers to destroy Base B, then hypothesize the existence of the three bombers that the fighter is escorting.

```
IF object ID is fighter
     AND mission is escort
     AND fighters are above escorted planes
     AND fighters are not forward of escorted planes
THEN formation is not near its target
```

IF object ID is fighter
AND mission is escort
AND fighters are moving forward of formation
THEN formation is approaching the target

The above two rules tell what maneuvers fighters make when they are escorting other planes. When the formation is cruising, and not near the planned target, fighters weave and execute "S-turns" so they do not overtake their partners. However, when the formation nears the target, the fighters fan out in front of the formation to intercept any opposing planes.

IF aircraft "pops up"
AND heading is along LOS[25] of possible target
THEN weapon delivery on target is imminent

This rule says that terrain following aircraft will pop up and point towards their target just prior to attack.

IF an object's munitions is bombs
THEN it has no rockets

IF an object's munitions is rockets
THEN it has no bombs

Most all-purpose fighters can carry bombs or air-to-air missiles, but not both.

Mission-Oriented Rules

These rules deal with the inference of opposition's mission from maneuvers. Most planes fly a pre-planned mission with

---

25. LOS: Line-Of-Site

phases like take-off, ascent, cruise, dash, attack, retreat, cruise, and landing. Each phase has its own characteristics: ascent has high rate-of-climb, dash has high acceleration, cruise has stable velocity and altitude, and attack has quick maneuvers. With determination of an enemy object's current mission phase, we may be able to infer time and distance until later phases.

```
IF altitude is very low
     AND object is close to FEBA[26]
THEN mission is close air support

IF maneuvers are erratic
     AND altitude is low
THEN mission is close air support

IF object has no look-down radar capability
THEN mission is not strike defense
```

"Look-down" refers to a radar's ability to distinguish moving objects against a cluttered background like the earth's surface. Radars can "see" much more easily against a cold, uncluttered background like the sky. Strike defense aircraft need this capability to be able to fly above planes that are interdicting.

```
IF a strike formation is approaching FEBA
THEN look for defense suppression aircraft

IF formation altitude is low
     AND maneuvers is not terrain-following
THEN formation is preparing to attack
```

---

26. FEBA: Forward Edge of Battle Area, the imaginary dividing line between friendly and opposing forces.

```
IF formation altitude is low
     AND velocity is slow
THEN formation has limited self-defense capability

IF maneuvers is S-turns
THEN mission is escort

IF maneuvers is terrain-following
THEN mission is bombing-run
IF formation mission is cruise
     AND new object (OBJ_NEW) joins formation
     AND dist(OBJ_NEW, OBJ_ANY) is very small
THEN mission(OBJ_NEW) is refueling
     AND formation is bombers or transports
     AND look for NEW_OBJ to soon leave formation
```

Cyclical Rules

This set of rules exhibits "chaining", whereby the antecedent of one rule becomes true (the rule fires) because the consequent of another rule fired at an earlier time. These rules work as follows: if the identity of a plane is known, its combat radius (how far it can fly between refuelings) is also known. Given the combat radius and the plane's current position, and assuming enemy base positions are known, all possible origins for the plane can be determined. Furthermore, pre-mission briefings might have yielded deployment information (percent of each kind of plane at each base) about the enemy bases. This information, finally, "feeds back" on ID, and serves to "bump up" or "bump down" the current estimate of the probability of ID.

```
IF object ID is fighter
THEN combat radius < 1000 nmi

IF object ID is bomber
THEN combat radius < 8000 nmi
```

```
IF object ID is transport
THEN combat radius < 6500 nmi

IF distance-to-base < (combat-radius / 2)
THEN base is a possible point-of-origin

IF object's point-of-origin is known
THEN examine deployment at that base and update ID
```

ECM Rules

Listed here are a few other rules that are not currently used in our expert systems because electronic counter-measures (ECM) have not been added to the environment model yet.

```
IF SOJ[27] is detected
THEN hypothesize a screened aircraft attacking
     a high-value target

IF SSJ[28] is detected
THEN hypothesize additional aircraft in formation
```

These rules say that the enemy uses electromagnetic jammers to screen planes of high value.

```
IF position of OLD_OBJ is not received
     AND time frame has ended
THEN estimate current position of OLD_OBJ
     by dead-reckoning
```

This rule says to estimate the current position of targets with no up-to-date sensor returns by extending the last-known heading of the plane.

---

27. SOJ: Stand-Off Jammer, a ground-based jammer that attempts to obscure air formations with electromagnetic emissions.

28. SSJ: Single-Side Jammer, an airborne jammer that flies with a formation to screen its approach.

Example Rules from Defense Systems Corp.

Included here are a couple of rules derived from interviewing pilots and recording their protocols. The rules are more tactics-oriented, but show the future development path of the object analysis expert system.

IF enemy aircraft turns worse and accelerates
better than own aircraft
THEN anticipate enemy initiating a climbing attack

IF enemy fighter formation splits equally left
and right
THEN anticipate attack from a pincer maneuver

CHAPTER 4
OAS1: A RULE-BASED EXPERT SYSTEM

Introduction

OAS1 was the first of two expert systems created as a result of this research. Implementation was begun as soon as a small portion of the knowledge engineering was complete, so the resulting system has many imperfections. We felt it was important, however, to begin coding as quickly as possible, even though the design was not perfect. We would never have accomplished as much as we did if we had delayed implementation until the knowledge engineering was "done". Knowledge engineering, after all, is never really finished. OAS1 was built as a learning tool, to be discarded after completion. The discussion below describes OAS1, what we learned from it, and how it influenced the design of OAS2.

Design Goals

The overall goal of OAS1 was to demonstrate the feasability of performing object analysis with an expert system. Design was completed with three constraints and sub-goals in mind: (1) use existing software tools, (2) include a belief-maintenance mechanism, (3) demonstrate as many rules as possible.

The first requirement, to use existing software tools, entailed using an Hughes-developed knowledge representation system called ARF [Edwards, 1984], A Rule and Frame system.

ARF is described in more detail below, but is basically a frame-based representation system like the one described in **Chapter 2**. One goal of OAS1 was to evaluate ARF for use in future expert system applications.

Requirement two, the inclusion of a belief-maintenance mechanism in OAS1, was considered important because of the need for determining what sort of belief mechanism object analysis would require. We felt it was important to evaluate one candidate belief mechanism (a Bayesian tree for OAS1) to see if it would be powerful and flexible enough for future implementations.

Given the constraint of using ARF's built-in rule constructor, the third design goal was to implement as many of the developed rules as possible. Of course, not all of the knowledge engineering discussed in **Chapter 3** had been completed at the time of OAS1's development, so a side-effect of the first-cut implementation was to determine those rules and knowledge areas that looked the most promising for object analysis.

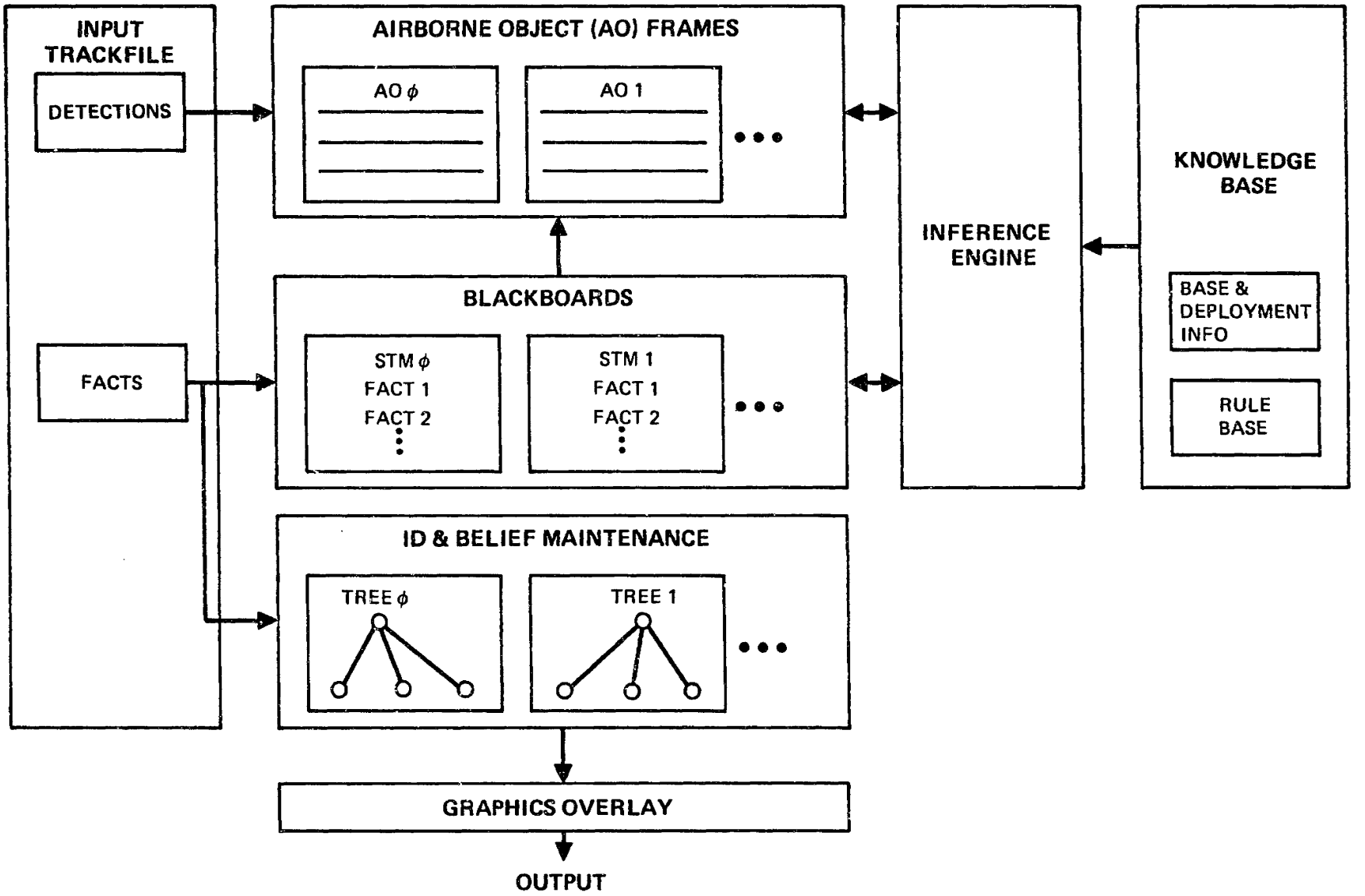
The next few sections examine the structure of OAS1, giving detailed descriptions of how the components interact. Following that are some sample runs and results, and a discussion of the effect that OAS1's development had on the design of OAS2.

Overview of OAS1

Figure 3 is a block diagram of the expert system, showing the five main components (**Airborne Object Frames, Blackboard, ID and Belief Maintenance, Inference Engine,** and **Knowledge Base**), input and output, and interactions between the blocks. The driver of the system is the input trackfile, which consists of time-sequenced ownship-centered simulated sensor returns from surrounding airborne objects. To construct a trackfile, a scenario is devised in which ownship faces a formation of incoming enemy aircraft. Given ownship's sensor suite, we then determine what information, if any, each sensor would provide on each object in the enemy fleet, and at what time and range it would provide it. The sequence of returns for each sensor is then assimilated into one composite file of time-ordered sensor information on all the airborne object tracks in the scenario. This ordering simulates how the expert system would receive sensor data in a real avionics suite, except that each sensor would have to have the ability to *asynchronously* report any newly-detected information on an airborne object.

Entries in the OAS1 input trackfile are either "detections" or "facts". Detections indicate when a new airborne_object is found, presumably by the longest-range sensor available, the radar. A detection causes three things to happen: first, an empty airborne object frame is created to hold information

FIGURE 3: OAS1 BLOCK DIAGRAM



about the new object. Second, an empty STM (short-term memory) is created and "posted" on the blackboard. This STM is used to hold facts about the airborne object. Third, a Bayesian tree structure is created to maintain the probability of the object's ID. (The belief mechanism is described in more detail below).

Input trackfile "facts" represent actual sensor returns from an airborne object. These facts can do one of two things: they can be placed on the STM associated with the airborne object for use by the inference engine in checking rules, or they can go directly to the belief mechanism for updating the identity of the airborne object.[29] (See the **Input Assumptions** section for some system simplifications which allowed us to ignore some real-world sensor problems.)

To summarize, for each new object detected, an airborne object frame is created to hold information about the object, an STM is created to hold sensor data about the object, and a Bayesian probability tree is created to keep track of the object's ID. The input trackfile then puts the sensor returns

---

29. For efficiency reasons in OAS1, a fact's destination (either the inference net or an STM) is **predetermined** in a trackfile; an operational system should have a "preprocessor" determine the appropriate place to send facts.

on previously detected objects onto their STMs and belief trees. At the end of a fixed length of time (nominally 30 seconds), the inference engine cycles through the rule base, trying to satisfy antecedants by looking at frames, STMs, belief trees, or the knowledge base, which contains information like plane-deployments at bases. For each antecedent that tests true, the rule "fires", and the consequent of the rule either adds information to the airborne object's frame or STM, or updates the object's ID by passing data to the belief tree. The inference engine cycles through the rule base until a pass produces no new information, and then the input trackfile resumes sensor data assertion and airborne object detection. Facts that cause a rule to fire are removed from the STM at the end of a cycle; otherwise, they remain until used.

The output of all this is a vector that represents the probability of ID for each possible type of airborne object allowed. For our initial system, we decided to allow only six different kinds of airborne objects to be identified: MiG-29 and Su-27 fighters, Tu-95 bombers, Tu-144 transports, and AA-7 and AA-9 rockets.[30] Therefore, the output of the system for each detected airborne object is a six-element vector indicating the probability that the object is a MiG-29, Su-27,

---

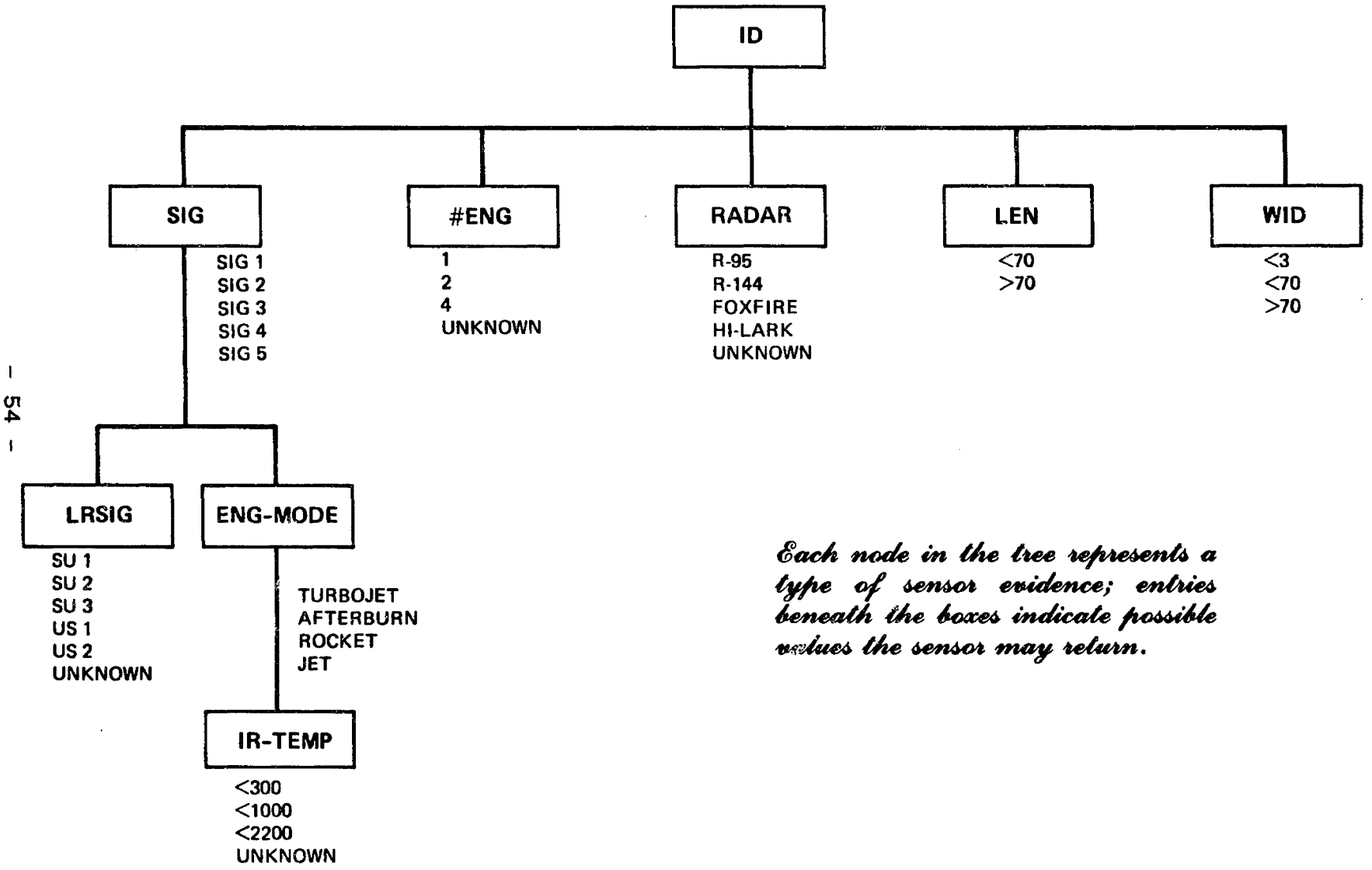
30. The system is in no way constrained to six objects; this small number was chosen to simplify things at the beginning.

Tu-95, etc. The initial probability distribution for a newly-detected target is (.167 .167 .167 .167 .167 .167), because each of the six IDs is equally likely. Following is a more detailed discussion of how the expert system maintains belief.

Belief Maintenance

The belief maintenance architecture used in OAS1 is a hierarchical Bayesian inference net based on the theory of Pearl [Pearl, 1982], and implemented with modifications by Mark Young of Hughes. The goal of belief maintenance is to compute the probability of a hypothesis based on evidence, and to be able to perform updates on the hypothesis whenever new evidence is received. In object analysis, the hypothesis is the ID of the airborne object and the evidence is sensor returns. The hierarchical arrangement of object analysis evidence is shown in **Figure 4**. ID, the goal of the inferencing, is located at the "root" node of the tree. One level below are sensor returns that directly affect ID: signature (SIG), number of engines (ENG), radar type (RADAR), length (LEN), and width (WID). "Direct effect on ID" means we are able to assign a value to the conditional probability, $Prob(S_{i,j}|ID_k)$, for all i, j, and k, where, for **Figure 4**, i indicates one of five possible sensors (SIG, ENG, RADAR, LENGTH, or WIDTH), j indicates one of the possible **values** that sensor S_i can have, and k indicates one of the (six) possible IDs. In other words, if the RADAR sensor can have the possible values of "R-95",

FIGURE 4: SENSOR EVIDENCIAL HIERARCHY



Each node in the tree represents a type of sensor evidence; entries beneath the boxes indicate possible values the sensor may return.

"R-144", "Foxfire", "Hi-Lark", and "Unknown" (j goes from 1 to 5), we are able to write a probability matrix:

R-95	R-144	Foxfire	Hi-Lark	Unknown	
0.75	0.20	0.01	0.01	0.03	Tu-95
0.15	0.80	0.01	0.01	0.03	Tu-144
0.00	0.00	0.90	0.10	0.00	MiG-29
0.00	0.00	0.15	0.85	0.00	Su-27
0.05	0.05	0.00	0.00	0.90	AA-9
0.05	0.05	0.00	0.00	0.90	AA-7

This matrix completely describes the effect that radar returns have on ID. For instance, if the ID is given to be a MiG-29, then according to the matrix above, Prob(radar is Hi-Lark) = .10 and Prob(radar is Foxfire) = .90.

Looking at **Figure 4** again, the long-range signature (LRSIG)[31] and engine mode (ENG_MODE) sensor returns have an effect on SIG, and infrared temperature (IR_TEMP) affects ENG_MODE. We can write conditional probability matrices for these relationships in a manner similar to that shown above, yielding $Prob(S_{l,m}|S_{n,p})$ matrices that show, for instance, the probability that IR_TEMP is less than 300 degrees, given that ENG_MODE is afterburn.

---

31. LRSIG is a parameter that is representative of the ID-specific information that would be received about an airborne object at long range.

Given the $Prob(S_{i,j}|ID_k)$ and $Prob(S_{l,m}|S_{n,p})$ matrices derived through knowledge engineering, we must now find $Prob(ID_k|S_{r,s})$, the probability that ID is one of k possible values, given some new piece of sensor evidence--that sensor S_r has value s.[32] Bayes rule, which allows computation of Prob(H|E) by:

$$(4.1)\quad Prob(H|E) = \frac{Prob(E|H)\ Prob(H)}{Prob(E)}\ ;(H:hypothesis,\ E:evidence)$$

provides the type of solution needed, but it has several shortcomings for our implementation.

First, there is an instability problem in updating the tree: if an increase in the confidence of evidence causes increased certainty in the hypothesis it supports, and, in turn, stronger belief in that hypothesis means greater expectation for occurance of the evidence, the update may continue forever.[33,34] This occurs because the update of the "father" node depends on the prior probability of the "son" (as shown by Bayes rule), and likewise, the update of a son node depends on the priors of its father (because P(E) = Prob(H) Prob(E|H)).

---

32. For **Figure 4**, k = 1 to 6, and r = 1 to 8.

33. See [Pearl, 1982, p.3]

34. It doesn't matter whether the update is an increase or a decrease in confidence.

A second problem is the Bayes rule requirement that the prior probability, Prob(H), be known to compute Prob(H|E). However, in an evidencial network, each node acts as both evidence to support nodes above it, and as a hypothesis to which nodes below contribute evidence. There is thus a problem in knowing the priors for an intermediate node in the network, due in part to the subjectivity and inconsistency of expert information [Duda, Hart, Nilsson, 1976, p.5].

The final and most severe limitation of a straight Bayes rule implementation is that it only allows binary-valued hypothesis variables to be used. For example, in calculating Prob(H|E), H and $\bar{H}$[35] could only be "ID is fighter" and "ID is not fighter", respectively. What we really want for object analysis is for H to be multi-valued: H_1 is "ID is fighter", H_2 is "ID is bomber", H_3 is "ID is rocket", etc., so we could calculate Prob(H_i|E) for all values of i.

Pearl's solution is an algorithm, consistent with the theory of Bayes, that (1) constrains diffusion of new evidence into the tree to a single pass, (2) eliminates the requirement of knowledge of prior probabilties,[36] and (3) allows the use of

---

35. $\bar{H}$, pronounced "H-bar", means the opposite of H, or **not** H.

36. except at the root node, as discussed below

multi-valued variables. If we want to find the likelihood of the states of some node A induced by some data D, part of which, $D^u(A)$, comes from above A and part of which, $D_d(A)$, comes from below, then according to Pearl, we use:

(4.2) $Prob(A_i|D^u(A),D_d(A)) = \alpha * Prob[D_d(A)|A_i]*Prob[A_i|D^u(A)]$[37]

where α is a normalization constant. This is analagous to Bayes rule, when (4.1) is written in a different form:

(4.3) $O(H|E) = \lambda(E) * O(H)$

where $\lambda(E)=Prob(E|H)/Prob(E|\bar{H})$ is the **likelihood ratio**, and $O(H)=Prob(H)/Prob(\bar{H})$ is the **prior odds**.

Pearl's generalization allows the update of the probability of *any* node in the tree, since the prior odds term $[Prob(H)/Prob(\bar{H})]$ in (4.3) has been replaced in (4.2) by $Prob[A_i|D^u(A)]$, which means *the likelihood of a variable state, given all the evidence gathered by the network above it.*[38] This allows the update of an intermediate node in the attribute hierarchy to take place in a single pass, since that node need

---

37. From [Pearl, 1982, p.6].

38. Paraphrased from [Pearl, 1982, p.6].

only "communicate" with its immediate father and son nodes. Equation (4.2) separates the dependence of A_i upon *both* $D^U(A)$ and $D_d(A)$ into two multiplicative terms, each of which includes only $D^U(A)$ *or* $D_d(A)$ factors. See page 16 of [Pearl, 1982] for implementation details of (4.2).

The replacement of "prior probabilities" by "evidence above the node in question" discussed in the last paragraph works at every node except the root node, since it has no network above it. For this special case, $D^U(A)$ should be interpreted as "available background knowledge", rendering it identical to the classical notion of subjective prior probability. For our expert system, upon creation of a new network structure, this root node background probability is distributed equally among the possible IDs. With six possible objects, the initial probability vector looks like (.167 .167 .167 .167 .167 .167).

To summarize: sensors, which are assumed independent,[39] "assert" the returns they get from airborne objects into the appropriate inference net node[40] with some confidence level between 0.0 and 1.0 that depends on range, weather conditions, time on target, etc. The change in belief at the node

---

39. Each sensor asserts its belief without knowledge of what any other sensor asserts; there is no correlation between them.

40. For **Figure 4**, there are eight possible nodes.

propagates up the tree to the ID root node, and then down the remaining branches to the lowest leaves, updating likelihood vectors along the way. The complete update of the tree is finished in only one "cycle", with each node visited just once.

As indicated in the **Overview** above, not all the evidence used to maintain belief comes directly from the input trackfile. The rule base and inference engine provide an alternate tool to perform more complex deduction than that possible with the inference net alone. Handling the combined effect of velocity and radar cross-section upon ID, for example, is much easier with rules. In general, knowledge engineering that resulted in rules with simple if-parts and consequents that directly affect ID were embedded in the inference net; those with complex if-parts or "chained" inferencing (like the cyclical rule set) were implemented with rules. We also felt that rules used as an alternate evidencial reasoning tool would give the system flexibility and extensibility: it wouldn't always be necessary to redesign the sensor hierarchy or probability matrices--code for rules would only need to be added.

Since the inference net is the only mechanism that actually maintains ID probability vectors, rules must have some way of making assertions to the inference net, just as trackfile sensor facts can. A modification to the Bayesian inference net

allows this: assertions about ID can be made directly to the root node. To illustrate, one rule discussed above was:

```
IF velocity > Mach 1.5
AND radar cross-section is medium
THEN object ID is fighter
```

If the trackfile asserted onto the STM the velocity and radar cross-section facts that satisfy this rule, the rule would fire and the inference engine would assert "ID is fighter" into the net with a confidence that is a function of the "belief" in the above rule, and the confidence that "velocity > Mach 1.5" and "radar cross-section is medium".

Suppose the rule fired and asserted, "ID is fighter, confidence = 0.8". To update ID, an **assertion vector**, $\vec{e}$, would be created, with 0.8 probability distributed equally over IDs that were fighters, and (1.0 - 0.8) = 0.2 probability distributed over the remaining IDs. For our case,

$$\vec{e} = [.05 \ .05 \ .40 \ .40 \ .05 \ .05]$$

since our vector of possible IDs is

$$\mathbf{ID_{poss}} = (\text{Tu-95 Tu-144 MiG-29 Su-27 AA-9 AA-7}),$$

and MiG-29s and Su-27s are both fighters. The vector $\vec{e}$ is combined with the current **background probability vector**,

$\mathbf{Prob(ID)}_{curr}$, of $\mathbf{ID}_{poss}$ according to the formula,

$$\mathbf{Prob(ID)}_{i,new} = \frac{\mathbf{Prob(ID)}_{i,curr} * \vec{e}_i}{\sum_j \mathbf{Prob(ID)}_{j,curr} * \vec{e}_j} \quad ,^{41}$$

to calculate the i^{th} element of the new background probability vector, $\mathbf{Prob(ID)}_{i,new}$, for the root (ID) node of the inference net.[42] This formula says to multiply the assertion vector by the background probability vector element-by-element, and renormalize the result so the probabilities sum to unity. The background probability vector is used by the inference net in the computation of the **actual** probability of ID vector that is the output of the expert system. As mentioned above, when a new inference net is created for an airborne object,

$$\mathbf{Prob(ID)}_{curr} = [.167 \ .167 \ .167 \ .167 \ .167 \ .167] \ ,$$

indicating that all IDs are equally likely. By the method outlined above, rules that make assertions directly about ID

---

41. $\mathbf{A}_i$ denotes the i^{th} element of the vector **A**, and the operator * is product.

42. $\mathbf{A}_i$ denotes the i^{th} element of the vector **A**, and the operator * is product.

have an interface to the belief-maintenance mechanism.

To insure that multiple firings of the same rule do not adversely alter ID, a list of rule firings and their arguments (assertion vectors) is kept. In this way, if a sensor provides information that is not new, but still causes a rule to fire, that rule will be ignored. Similarly, if Rule i asserts "ID is fighter, confidence =.7" at time j, and then asserts "ID is bomber, confidence = .5" at time j+1, the effect of the first assertion will be removed from the net. We therefore assume that the most recent assertion of a sensor or a rule is also the most accurate.[43]

For rules to assert results to the net *with* confidence levels, the inference engine must have a way to propagate confidence from the facts in the trackfile to the results of a rule's consequent. This is discussed in the next section, which describes OAS1's knowledge representation system in more detail.

ARF: Rules and Frames

ARF is a frame-based knowledge representation system that features an inference engine and rule constructors. Among the

---

43. This, of course, is not true when the airborne object is moving away from ownship. The system does not handle this case.

pre-defined frame objects in ARF are STMs, FACTs, and RULEs. STMs (or blackboards) are special frames upon which FACTs can be instantiated with confidence values. FACTs are specialized frames which have object,[44] attribute, and value slots. RULEs are generalized "if-then" constructs with an associated confidence level, and include special macros like "check-fact" and "object-test" to facilitate pattern-matching. In addition, RULEs can use wildcarding, so **(check-fact '* ' eng 2)** in the if-part of a rule will find any frame object with the attribute "engines" and value "2" (a plane with 2 engines).

The inference engine is a separate entity in ARF, and, when invoked, cycles through the rule base comparing if-parts with FACT frames and executing then-parts when appropriate, until no rules fire during a complete pass of the engine.

Confidence propagation is achieved in the following manner. When a rule is ready to fire, the inference engine multiplies the confidence associated with the rule by the product of the confidences of each FACT checked in the if-part. This result becomes the confidence level with which the then-part is asserted. The result of executing a then-part is either a

---

44. Here, object refers to any frame instance in the system, such as instances of airborne-object frames.

FACT-assertion back onto the STM,[45] or an ID-assertion to the inference net.

Besides their use for STMs, FACTs, and RULEs, frames are used in the object analysis expert system for holding long-term information about airborne objects, such as position, mission, and mission-phase. These airborne-object frames are more accessible than STMs, which are only used during inference engine operation. Originally, frames were also meant to hold formation information about airborne objects--which objects belonged to which formation, what was the heading or possible target of the formation, etc. This feature was not implemented in OAS1 because development work was halted once we saw problems with its design. Formation-level and target information, as well as maneuver information is used, however, in the second expert system, OAS2.

Input Assumptions

As mentioned above, we have ignored several real-world problems and made several important assumptions in constructing our trackfiles of simulated sensor returns. First, we have constrained the trackfile to contain only sensor data that comes from ownship. There is no "cooperation" from other

---

45. These assertions can either change the **value** of a FACT that already exists, or create a new FACT.

planes or external sources of information. Incorporation of alternate knowledge sources is left for the future. Second, each sensor is assumed to have some sort of pre-processor that performs decision-making, resolving possible ambiguity in a sensor's returns. We've also assumed the sensor processor assigns confidence levels to the data. A third assumption is that sensors have no correlation problem: they have a perfect ability to assign a sensor return to the *correct* airborne object, even though there may be many objects located within a small airspace. This assumption, of course, eliminates a serious problem a real sensor-driven object analysis system would have. Our fourth and final assumption is that sensors have an unlimited amount of time to "look" at any airborne object desired (thus producing the best possible information given the range), and that there is enough time to use all the sensors (if desired) to examine an object.

Naturally, in a real system, time is a very valuable resource. We could overcome this problem, however, by utility analysis. One possible scheme would be to prioritize rules (and therefore the sensors they use) by calculating the quotient **{rule cost/ID importance}**, where rule cost is a function of the sensors needed to check the rule and ID importance is an estimation of the object's threat value. For an initial development effort, we have chosen to ignore these four problems. A treatment of utility analysis is presented in

[Blackman and Broida, 1983]. Other computational considerations for sensor fusion are considered in [Waltz, 1981].

A Sample Run

A sample run of OAS1 is shown in **Appendix A**. The scenario depicted is ownship facing two incoming planes--one MiG-29 and one Su-27 fighter. A commented listing of the trackfile that drives the system is shown, along with the corresponding effect that each sensor return has on system output. The trackfile for the scenario was purposely constructed with misleading long-range sensor returns to see how the system would recover from sensor inaccuracy. See the appendix for more details.

The normal output of the system is a time-history of the probability of ID vectors as sensor data accumulates. For demonstration purposes, a graphics front-end was added to OAS1. Planes appear on the screen as they are detected by sensors, and transitions in ID are depicted by changing graphics icons. Two sample screens from the scenario are included in the appendix.

System Evaluation

For a first cut at an expert system in a new knowledge domain, OAS1 functions reasonably well. It is able to take a

sequence of simulated sensor returns and accurately identify airborne objects, although not always in the most robust or efficient manner. The flaws in OAS1's design were caused primarily by the constraint to use the software tool ARF, and because implementation and design occurred simultaneously, for the most part. As mentioned in **Chapter 1**, however, we thought it best to begin implementation quickly, to facilitate the iterative nature of system design and knowledge engineering. We viewed the first object analysis expert system primarily as a learning tool. The shortcomings of the system will be discussed momentarily, but first let us focus on OAS1's positive aspects.

A strength of OAS1 is the manner in which the Bayesian inference net handles belief maintenance. As shown by the sample run in **Appendix A**, the ID vectors make transitions commensurate with the confidence in the facts that cause the changes: facts with little confidence cause small changes in Prob(ID), and vice-versa. The inference net also allows hierarchical structuring of sensor attributes. It is easy to see sensor dependencies from tree structures like **Figure 4**, and sensor relationships are clearly determined by probability matrices. The ability of the inference net to make smooth transitions in ID is apparent from the sample run: the net accurately followed the initial assertions that the objects were bombers and transports, but gradually reversed itself as

sensor data began to indicate that ID was really fighters. It was encouraging to see that the inference net did not make any sudden "jumps" in ID probability. In addition, the method we chose of interfacing the rule base to the belief mechanism through assertions to the root node seemed to work well. Upon examining the sample trace in the appendix, we see that the rules help to correctly identify the objects sooner. OAS1 also demonstrates that two separate inferencing mechanisms (the inference net and the rule base) can operate independently, yet both contribute towards determining ID.

Another positive result of OAS1 is that it helped to identify promising rule areas. The cyclical rule set, in which ID implies combat radius, combat radius implies point-of-origin, and point-of-origin implies ID, appears to work effectively in OAS1, allowing correct ID to be determined more quickly than without the rule set. In general, using rules that "post" facts to be used by other rules works well within ARF's framework. In addition, we've confirmed that the ability to trigger new rules from old rules (by "chaining") is required for the object analysis task.

Shortcomings

OAS1's principle weaknesses are a result of using ARF's "packaged" facility for rules and inferencing. Overall, there is little flexibility in choosing how knowledge, rules, and

confidence propagation interact. This rigidity has kept us from using all the rules we developed and has led to a number of inefficiencies.

One problem is the lack of intelligent coupling between the reception of new sensor evidence and the testing and execution of rules. With the ARF inference engine, there is neither a way to limit which rules are tested, nor which object frames they are tested upon. As soon as the inference engine is invoked, *all* rules are tested on *all* airborne-object frames until no rules fire during a complete pass through the rule set. In contrast, we shall see that OAS2 allows rules to be tested singly upon any airborne object.

Another problem is the lack of an easy interface to confidence computation. If, for example, the result of a rule-firing is the assertion of a new fact onto an STM ("chaining"), we have no way to easily access either that fact or its confidence factor for use in other parts of the system. Furthermore, the confidence propagation method of OAS1 is fixed by the inference engine; this is no way to experiment with different methods of computing belief factors.

In the future, we want to eventually write rules that do more than just check simple attributes of an object. We want

an inference engine and rule base that allows mission-level inferencing, message-passing to the pilot, prioritization of rule testing, and checking for the occurrence of long-term events, such as plane maneuvers or radar scan patterns. We want to vary the method of confidence propagation as it proceeds from sensor evidence to rule consequents. This would allow the evaluation of different interfaces to the belief-maintenance mechanism. In short, OAS1's limited rule-construction ability will restrict the power and flexibility of the expert system as the problem domain grows.

OAS1's use of the Bayesian inference net causes some extensibility and efficiency problems, although they might exist in any expert system that maintains belief. Expansion of OAS1 to handle either more sensor inputs or a greater number of possible IDs is not easy. To perform either extension, each probability matrix[46] that relates sensor evidence to identity must be completely recreated, so that the dimensions are correct and all probabilities sum to one. Furthermore, there is no easy way to tell the system to "forget" about a possible ID for a sample scenario, in case we wanted to evaluate system performance for a run where we knew there would be no planes of

---

46. A sample matrix was shown in the **Belief Maintenance** section of this chapter.

that ID used.

An efficiency concern of the Bayesian belief-maintenance mechanism is that the inference net **must** be updated *every* time new sensor evidence is received. This update process is fairly time-consuming, even though (1) we are using a relatively small sensor attribute tree (**Figure 4**), and (2) we are using Pearl's method to limit the update to a single pass. We can probably eliminate these inefficiencies by using Shafer-Dempster theory to maintain belief, since it requires updates of the tree to be done only when Prob(ID) is needed.

The result of the structural shortcomings of OAS1, and to a lesser extent, the Bayesian belief mechanism, is a system that is annoyingly slow. Though not the most important weakness of the system, it is certainly the most frustrating. Despite the use of only two airborne objects, six possible IDs, and compiled code, the sample run of **Appendix A** takes over five times real time, using thirty-second frames. While speed is of little concern in the design of prototype AI systems, OAS1 is slow enough to impair the system's development cycle. It is very difficult to evaluate the usefulness of rules, assign belief ratings to rules, and examine tradeoffs in knowledge representation when it takes so long to see the effect of making a single change.

In addition, OAS1 is space-inefficient. This is primarily because the development work is being completed on engineering workstations (Apollo DN300s) running LISP, rather than on dedicated LISP machines. Because the Apollos were not designed specifically for executing LISP code, the space allocated for heaps (storage) is fairly small. As a result, the machine garbage-collects (reclaims unused memory cells) quite often, halting the expert system's operation. It appears, though, that OAS1's space-inefficiency is caused primarily by ARF, since OAS2, without ARF's inference engine and rules, uses only one-eighth the space of OAS1.

Two final concerns with OAS1's belief mechanism are that, unlike Shafer-Dempster theory, Bayesian inferencing (1) has problems maintaining consistency in degrees of belief for evidence that represents ignorance [Shafer, 1976, p.24], and (2) does not distinguish between evidence that comes from a single source and evidence that comes from disparate sources [Lowrance and Garvey, 1983, pp. 13-15]. In reference to the first concern, the justification for using a belief-maintenance mechanism that has the ability to represent ignorance is that the likelihood of erroneous evidencial reports is reduced. To quote Lowrance and Garvey:

> "The primary advantage of this approach is that each knowledge source can express itself at a level of detail of its own choosing. When there is no clear

> reason to prefer one proposition to another, that judgment can be suspended. Thus, a radar operator can express some belief that an object is at a given location without having to speculate as to that object's type. A Bayesian approach would require that a precise probability be assigned to each type, no matter how noisy the sensory data, and no matter how little statistical data are available from which to make justifiable estimates."[47]

In object analysis, assertions with ignorance are of the form, "ID is *not* bomber, conf = n". The ability to use rules involving assertions such as this should become much more valuable as further rule-research is accomplished. Furthermore, the Shafer-Dempster ability to integrate disparate sources of evidence will become important as we add more knowledge sources (sensors) to the system.

Implications for the Development Effort

Since the major shortcomings of OAS1 stemmed from the inference engine and rule base, a logical step was to implement them another way. For OAS2, the second cut at the object analysis expert system, we decided to keep the frame representation for knowledge, but implement rules using demons. In addition, we decided to forgo integration of a sophisticated belief-maintenance mechanism into OAS2; rather, we concentrated on making the design modular enough so its inclusion would

---

47. From [Lowrance and Garvey, 1983, p.13].

later be easy. We also started a development effort parallel to that of OAS2 to implement Shafer-Dempster theory, because of the advantages it seems to have over Bayesian belief maintenance. (These topics are discussed further in the next chapter.)

In conclusion, two lesser implications of OAS1's space and time inefficiencies were: (1) develop an alternate frame-based knowledge representation system similar to ARF, but without the special features needed for rules (this was developed by Craig Lee of Hughes while OAS2 was being constructed), and (2) use dedicated LISP machines for future work (not for OAS2, unfortunately), to provide increased execution speed and better storage management.

CHAPTER 5
OAS2: A DEMON-BASED EXPERT SYSTEM

Overview

The evaluation of OAS1 discussed at the end of **Chapter 4** served as a guide to the design of OAS2. Indications were that the frame system for representing knowledge was adequate for the object analysis task, but that the rule and inference engine combination caused structural and flexibility problems. The major shortcomings of the system were: (1) an inability to use all the rules developed because of the knowledge structure chosen, (2) the rigidity of the inference engine and its confidence propagation mechanism, which prevented investigation of different methods of interfacing with the belief mechanism, and (3) a restriction on the extensibility of the system, both in possible object identities and sensor returns. In general, OAS1's design relied too heavily on the pre-packaged ARF environment. We felt the expert system's design became constrained by the way ARF's rules, knowledge, and belief-maintenance interact.

In light of these results, our goal for OAS2 was to simplify the interaction between system components. Above all, we wanted to maintain the ability to alter the behavior and communication between different parts of the system to allow easy testing of different configurations. We felt this would result in a modular, flexible system. In turn, we felt a

modular design would promote system extensibility, and important feature considering the large number of possible IDs and sources of classification information in object analysis. Whereas the principle design goals of OAS1 were to evaluate ARF and a candidate belief system, and to "size-up" the object analysis task, the goal of OAS2 was to design a flexible system structure that would encourage evaluation of different rules, knowledge sources, and belief mechanisms.

Along the way, OAS2 acquired capabilities not previously included in OAS1 because of structural design limitations. These new features include an ability to provide requests for unavailable sensor information, group planes into formations, and determine possible targets of formations. (See **Appendix B** for a sample run of the system.) In addition, the rule base of OAS2 was extended to use more of the rules in **Chapter 3**, including mission-level inferencing. Overall, the system exhibits substantially improved performance, as shown in the Summary section later in the chapter.

The principle design change from OAS1 to OAS2 was the replacement of ARF's inference engine with one based on demons [Rieger and Small, 1979] [Dyer, 1983]. Demons allow coding of rules as general (LISP) procedures, resulting in far more flexibility in the use of knowledge engineering results. In addition, demons enhance system efficiency, since their built-

in control structure allows **selective** test and execution of demon-based rules, unlike the inference engine implemented in ARF. (Demons are discussed in greater detail below.) OAS2 also expands the use of frames to represent formations and formation-level information, and airborne objects in greater detail. In addition, the ARF feature of slot **procedural attachments** [Bobrow and Winograd, 1977] is used to trigger demons upon the arrival of relevant sensor information.

Further reduction in OAS2 of the dependence on ARF was accomplished through the elimination of the use of frames associated with the inference engine, such as those for STMs, FACTs, and RULEs. Instead, airborne-object frames centralize almost all knowledge represented in OAS2.

Another difference in OAS2 is the reduced emphasis on belief-maintenance. Rather than trying to determine the "best" belief mechanism for object analysis, and designing OAS2 around it, we decided to decentralize the design in such a way that that *any* belief-maintenance mechanism could be integrated into the expert system with little modification. In this way, candidate belief methods could later be evaluated with a working expert system.

As mentioned at the end of **Chapter 4**, one belief mechanism we currently have under development is based on the theory of

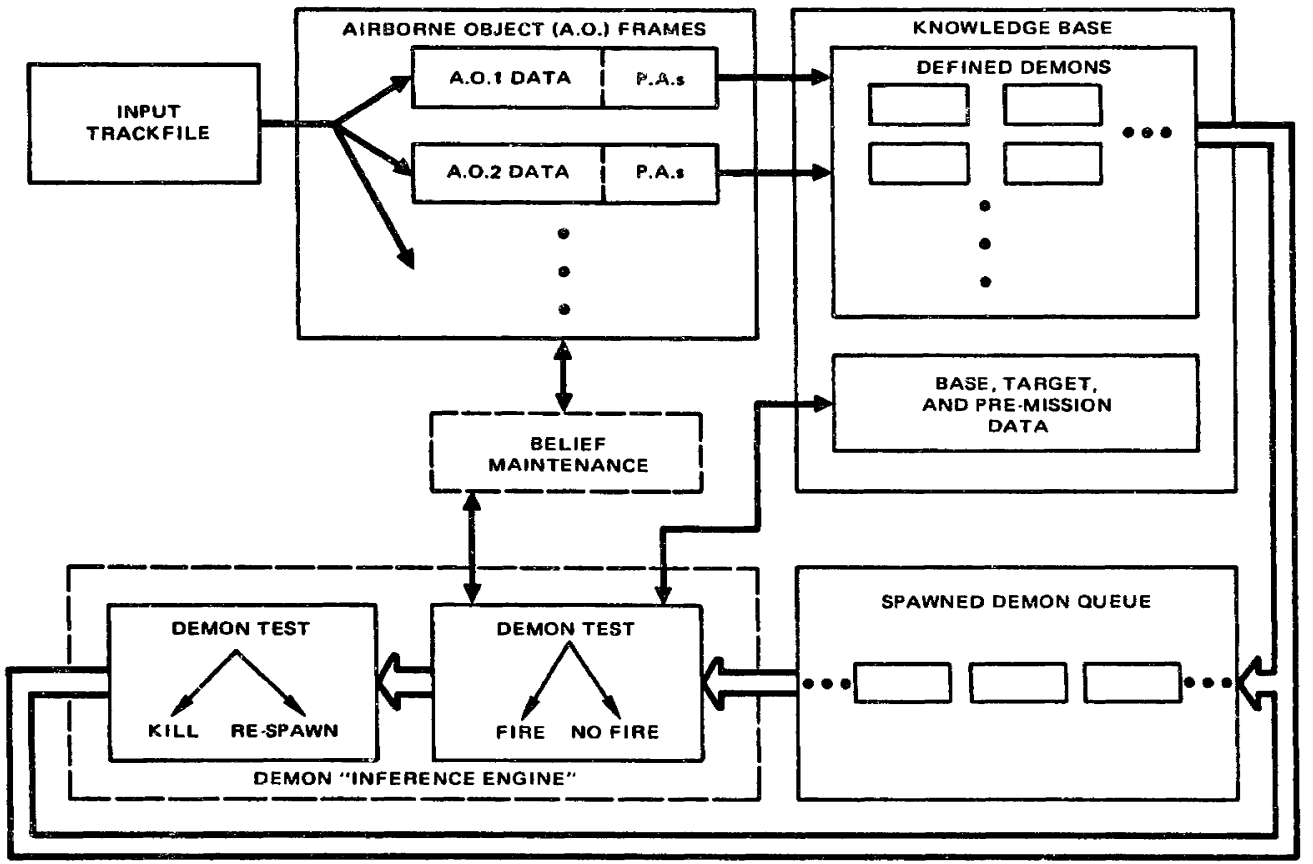
Shafer [Shafer, 1976] and Dempster [Dempster, 1967]. Shafer-Dempster theory is a generalization of Bayesian inference with the added capability of handling multiple evidencial reasoning for a large number of possible hypotheses. Mark Young of Hughes is implementing the theory in a tree structure similar to that of the Bayesian net, using the structure proposed by Lowrance and Garvey [Lowrance and Garvey, 1983]. Previous implementations of Shafer-Dempster theory serving as a guide include: [Strat, 1984], [Ginsberg, 1984], [Dillard, 1983], [Dillard, 1982], and [Garvey, Lowrance, and Fischler, 1981]. Eventually an evaluation to determine strengths and weaknesses of the two systems will be performed.

The remainder of this chapter examine the OAS2 design in more detail, and includes a description of the block diagram and background information on demons. This is followed by a sample run and an evaluation of the system. Some overlap discussion on elements common to both OAS1 and OAS2 is presented for completeness, even if it duplicates material in **Chapter 4**.

Design of OAS2

Figure 5 shows a block diagram of OAS2, with the major components of **Airborne Object Frames**, **Knowledge Base**, and **Demon-Control**, and input and output. The primary difference from OAS1 is the use of demons to implement rule-like behavior, and the expanded use of frames in knowledge representation.

FIGURE 5. OAS2 BLOCK DIAGRAM



The input trackfile has remained virtually unchanged,[48] and the knowledge base has been enlarged to include target and more formation-level information. The system design has a space for a belief mechanism, too, as shown by the dotted box. Currently, a rudimentary "time-smoothed" belief-maintenance mechanism is used, to be replaced later by the Bayesian inference net or Shafer-Dempster theory. Since there isn't a Input trackfiles are cumbersome to develop, for example, and probability-of-ID vectors are not presented to the user while the system is running. Because of OAS2's emphasis on knowledge centralization, however, it mechanism in OAS2, the output of the system no l. information is contained in airborne object frames, and must be viewed when the system is not running. This points out one weakness of OAS2--the lack of a sophisticated input and output capability. However, it should not be too difficult to add an I/O facility to OAS2 because of the emphasis on the centralization of knowledge.

Demons: Some Background

The design of OAS2 is centered about the use of **demons** for implementing rule and deduction capabilities. Demons have been previously used primarily in the natural language processing

---

48. The only difference is that trackfile entries are more English-like, and no confidence values are currently associated with sensor returns.

(NLP) area for expectation-based processing [Dyer, 1983]. Demons are used during the parsing of a sentence: as each word is "read", a demon that "knows" about this word and all its possible meanings "comes to life" and contributes to the understanding of the sentence. The nice thing about demons for NLP is that they're self-perpetuating: there doesn't need to be an underlying controller that determines which demons to bring alive and in what order to execute them. The demons figure it out themselves.

Demons are procedures with a built-in control mechanism to determine when and how often execution occurs. An unlimited number of demons can be created without seriously affecting the efficiency of the system because only those demons that have a chance of firing are ever processed. Until they are spawned, or "brought to life", demons stay in a dormant state, requiring no processing time. Once spawned, they are added to a queue called an **agenda** [Charniak, Riesbeck, and McDermott, 1980] or **working-memory-node** [Dyer, 1983], and when the appropriate "driver" code is invoked, they are retrieved (in the order they were spawned) to have their **test** code evaluated. If the test code evaluates to "true", the demon's **+act**[49] code is executed; otherwise, its -act[50] code is executed. This mechanism is an

---

49. Pronounced "plus-act", indicating the positive action to take.

50. Pronounced "minus-act", indicating the negative action to take.

exact analog to the **if-then-else** construct of a typical rule. A sample demon will help to illustrate the operation:

```
(demon velocity-RCS-demon
    (params mynode frame)
    (comment (test "if velocity > Mach 1.5 & RCS is medium
                    then ID is fighter"))
    (test (and ( >  (*fget* frame 'velocity) 1.5)
               (eq? (*fget* frame 'RCS) 'medium)))
    (+act (id-update frame 'fighter .9))
    (-act t) )
```

This piece of code defines **velocity-RCS-demon**, a demon for implementing the rule shown in the **comment** slot of the above definition. The **params** slot binds local variables to the parameters upon which this demon is spawned. For example, when we receive an update on the velocity of PLANE0, an airborne-object frame, and we want to test the above rule, we execute:

```
(spawn *wm* (velocity-RCS-demon PLANE0))
```

This adds the velocity-RCS-demon to an agenda called ***wm***, a working-memory-node that had been previously created. When the demon is ready to be **run**, the local variables **mynode** and **frame** are bound to ***wm*** and **PLANE0**, respectively, providing a way to access from inside the demon the arguments of previous demons (through other working-memory-nodes) and the knowledge stored in the **PLANE0** airborne object frame.

Upon invocation of the demon driver code, the **test** slot of **velocity-RCS-demon** is executed. For the code above, an ***fget***

(access) of the **velocity** and **RCS** slots of **frame** (bound to **PLANE0**) is performed to see if they are both *greater than 1.5* and *equal to medium*, respectively. If the result is "true", the code in the **+act** slot executes, asserting "ID is fighter, confidence = 0.9".[51] If the result of **test** is "false", the demon executes the code in the -act slot, returning true for the code shown above. We thus have an analogy to an if-then-else rule construct.

Demons continue to execute each time the driver-code is invoked, unless they are explicitly "killed", or removed from the agenda. This operation is part of the function of the driver code we mentioned above: as it takes each demon from the agenda, it checks its **fired?**[52] property, and if is equals true, indicating the demon has executed before, the driver kills the demon. Otherwise, the driver runs the demon. The one exception is that demons containing a "t" (true) in their -act slot kill themselves when they test false. This feature allows rules that should be tested only once to remove themselves from the demon queue, eliminating the need for an external control mechanism to do so.

---

51. "id-update" is the name of the procedure that maintains belief in OAS2.

52. A demon acquires the property, **fired?** = true, anytime its +act slot executes.

Rules built around a demon-oriented architecture feature tremendous flexibility and power. In addition to being able to test rule-like if-then-else constructs, demons can mimic the "chaining" and "fact-posting" ability of blackboards by using their +act slots to spawn other demons onto the agenda. The cyclical rule set,[53] as an example, is implemented by spawning a single demon that examines probability of ID and waits for it to cross an arbitrary threshold. When this happens, the **+act** part spawns a second demon whose task is to look at the object's combat radius, assign a probable point of origin, and spawn a third demon. This last demon examines deployment data of the possible bases and asserts additional information to identity.

This sequence demonstrates the simplicity of the interaction between demons, as they wait their turn, come to life, and execute their function, and then die. We have the option, though, of augmenting communication between demons by parameter-passing (through the **params** slot) and message-passing [Dolan, 1984], which allows primitive communication between demons. Though OAS2 does not currently use these features, we may come across rules in the future that could use a cooperative effort to solve an inference problem.

---

53. Described in **Chapter 3**--see the table of contents.

There are several other advantages to using demons for object analysis. Because they do not have to die after they execute, demons have the ability to "hang around" and wait for events to occur. For instance, one rule we developed requires detection of enemy "S-turn" maneuvers. We could spawn a demon to look for this trait for us, checking over several time frames to see if an airborne object's maneuvers oscillate transversely to its flight path.

OAS2 also uses **meta-demons** [Dolan, 1984], an extension that allows demons to be organized into logical execution groups. Meta-demons are "wrapped" around a set of constituent demons that are about to be spawned. The meta-demon then controls the execution sequence of its constituents. Three types exist: **OR-demons** execute when at least one constituent tests true, and then all constituents are allowed to execute at the same time; **AND-demons** execute when all constituent demons test true, and then all are allowed to execute at one time; **XOR-demons** allow the first of its constituent demons which tests true to execute, but none of the others.

Meta-demons are used in OAS2 to increase efficiency. For example, several groups of rules we developed are mutually exclusive, meaning only one of them can possibly execute. Wrapping an XOR-demon around the group when they are ready to spawn ensures that the first to test true will fire, but no

time will be wasted checking the others. Meta-demons also ensure that all demons in a group execute at the same, for the case when time-ordering of results is important.

Since demons afford us explicit control over which demons execute, we can afford to have a large number of *defined* demons (eventually the case for object analysis), while maintaining efficiency by running only those that have a chance at executing. Besides the ability to control *which* demons are run, we can also determine which airborne objects they are spawned upon. Unlike the rule set of OAS1, rules do not have to be tested on every detected airborne object.

Another advantage of demons is that we can modify the inference engine, or driver code. In the future, we can decide how to handle confidence propagation or evaluate different methods of interfacing with the belief-mechanism. We are not constrained, as in OAS1, to use a single method of processing data, rules, and belief. For example, one possible extension could be to prioritize the execution of demons, perhaps making sure that a demon that presents a message to the pilot is given immediate attention. Detection of a rocket coming towards ownship and the subsequent message that a demon might give a pilot, for example, should be given the highest priority. With flexibility in the inference engine, we should be able to fine-

tune the system as it changes and grows. Currently, this feature is used only to ensure that demons are killed once they execute the first time.

Finally, looking even further into the future, demons lend themselves easily to processing in a highly-parallel architecture. When a "burst" of new sensor data becomes available, a large number of associated demons are spawned. In many cases, the order in which they run is not important. In a parallel-processing environment, demons could queue in a list and wait for the next available processor. The processor would execute the demon, perhaps posting a message, updating belief, or spawning another demon into the queue. Demons that required time-ordered execution could be tagged so they were only accessed by the same processor, or were checked to see if it was their time to execute. A simpler scheme would be to allocate a processor for each new airborne object detected, which would handle all the demons associated with that object. Using this method, performance of the system would stay essentially the same no matter how many objects were detected. Of course, execution times of formation and target demons would increase with the number of airborne objects.

Airborne Object Frames

This section examines some of the other aspects of OAS2's design, including the knowledge representation and the use of

procedural attachments to control the execution of demons.

Referring again to the block diagram of **Figure 5**, defined demons reside in the **Knowledge Base**, along with pre-mission information and data on enemy bases and possible targets. One of the functions of the **Airborne Object Frames** is to spawn demons, moving them from their "dormant" state into the **Spawned Demon Queue**, where they can be accessed by the demon inference engine. Demon-spawning is controlled by procedural attachments (P.A.s) to the slots of airborne object frames; the slots hold sensor returns that are needed by the demon. When the input trackfile receives sensor returns on an airborne object, it puts the data into the appropriate slot on the frame, triggering a P.A. that spawns demons that use this data. To illustrate, a reduced version of the airborne object frame definition used in OAS2 is shown below:[54]

```
(defframe airborne_object
   (slot name)
   (slot velocity)
   (slot xy-pos)
   (slot heading)
   (slot RCS)
   (slot IR_temp)
   (slot radar)
   (slot maneuvers)
   (slot altitude)
   (slot prev-altitude)
   (slot hyp-ptr)
   (slot com_rad)
```

---

54. For a full-length example of an airborne object frame with slots filled, see the sample run of OAS2 in **Appendix B**.

```
    (slot base_POO)
    (slot poss_IDs)
    (slot ptr_list)
    (slot pID_list)

    (initialize (fput &node 'hyp-ptr
                      (make-instance 'hypotheses)))[55]

    (proc velocity  (spawn *wm*
                           (xor-demon (speed-demon1 &node)
                                      (speed-demon2 &node)
                                      (speed-demon3 &node)
                                      (speed-demon4 &node)))
                    put after)

*   (proc xy-pos    (set lastxy (fget &node 'xy-pos)) put before)

*   (proc xy-pos    (fput &node 'heading
*                         (cmpute-head (fget &node 'xy-pos)
*                                      lastxy))
*                   put after)

    (proc RCS       (spawn *wm*
                           (xor-demon (speed-demon1 &node)
                                      (speed-demon3 &node)
                                      (speed-demon4 &node)))
                    put after)

    (proc IR_temp   (spawn *wm*
                           (xor-demon (IR-demon1    &node)
                                      (IR-demon2    &node)
                                      (IR-demon3    &node)
                                      (IR-demon4    &node)))
                    put after)

    (proc altitude  (spawn *wm*       (alt-demon    &node))
                    put after)

    (proc com_rad   (spawn *wm*       (comrad-demon &node))
                    put after)

    (proc base_POO  (spawn *wm*       (POO-demon    &node))
                    put after)
)
```

---

55. The **initialize** piece of code executes upon instantiation of a frame; in this particular case, the code creates a pointer to a child **hypotheses** frame that keeps mission and phase information about its parent airborne object.

The above code creates an airborne object frame, which can be instantiated by executing,

(set PLANEn (make-instance airborne-object))

Values can be put into the slots of instance PLANEn by

(fput PLANEn slot-name value)

and can be accessed by

(fget PLANEn slot-name)

The code that creates slots in the above frame-definition is

(slot **slot-name**)

and the code to create a procedural attachment to a slot that has been defined is

(proc **slot-name** (... code ...) get/put after/before)

where the "/" indicates to choose one or the other. The get/put and after/before choices indicate whether the P.A. is triggered before or after a "get" or a "put" to the slot.[56]

Once within the executable code of the attachment, other values in the instance can be accessed with the variable "&node", which refers to the instance itself, and "&slot" which

---

56. Therefore, there are four combinations: before/get, before/put, after/get, and after/put. This is done so when the P.A. executes, it can have access to whatever state of the slot it requires.

refers to the slot-name of the current instance. Take, for example, the procedural attachments to the **xy-pos** slot, designated by asterisks on the previous page. The slot holds an ordered pair (x y), referring to the position of the airborne object relative to ownship. The two P.A.s compute the heading of the airborne object whenever new position information is entered into the frame. The first P.A. sets the global variable **lastxy** to the value of **xy-pos** *before* it is changed. The second P.A. computes the heading (using the **cmpute-head** function) with arguments of **lastxy** and the *new* **xy-pos** value,[57] and puts it into the **heading** slot of the airborne object.

The other P.A.s listed spawn demons using meta-demons. For example, the P.A. for **velocity** spawns four demons (rules) that have a chance of firing when new velocity information is received. The four are wrapped with an xor-demon meta-demon, to ensure that only the first of the demons to test true will execute. These two techniques promote efficiency: (1) only trackfile facts that are be used by a demon cause that demon to be spawned; and (2) meta-demons reduce the number of demons that are tested.

Most of the slots in the airborne object frame are fairly

---

57. The new **xy-pos** value was put into the slot between the two firings of the procedural attachments.

self-explanatory, except for the three slots dedicated to OAS2's belief mechanism: **poss_IDs**, **ptr_list**, and **pID_list**. An explanation of these slots can be found in the **Belief Maintenance** section, below.

Two other frame definitions are important for the knowledge structure of OAS2. These are the definition of the **hypotheses** frame, which contains mission-level information, and the **cluster** frame, which holds formation-level information. The definition of the former is:

```
(defframe hypotheses
    (slot mission)
    (slot phase)
    (slot acc-hist)
    (proc mission (announce-mission &node)  put after)
    (proc phase   (announce-phase   &node)  put after)
)
```

As mentioned in footnote 55 above, the **hypotheses** frame is created as a child to an **airborne-object** frame upon the latter's instantiation. A pointer to the **hypotheses** frame is put in the **hyp-ptr** slot of the parent airborne object. The **hypotheses** frame currently holds mission-level information about an object, including possible mission, phase, and targets. This data could, of course, be stored in the parent frame, but in the future there will be multiple hypotheses dealing with airborne object intent and function, so it is best to separate the knowledge now. When ECM capability is added to OAS2, the hypotheses frame may be used in rules such as, "IF a

stand-off jammer is detected, THEN hypothesize the existence of an enemy formation". Later, a separate belief mechanism just for hypotheses may be added to the system.

Cluster frames hold formation-level data, and are defined by:

```
(defframe cluster
    (slot name)
    (slot constituents)
    (slot heading)
    (slot targets)
    (proc constituents (fput &node 'heading
                             (find-avg-head (fget &node &slot))
                       ) put after)
    (proc heading (spawn *wm* (target-demon &node)) put after)
    (proc targets (spawn *wm* (sayfrm-demon &node)) put after)
)
```

These frames indicate how detected airborne objects are organized into groups. The **constituents** slot contains a list of pointers to airborne objects that are members of the formation, and the **targets** slot contains a list of possible friendly bases or planes that the formation may attack. The **heading** slot contains a value that is the average of the values in the **heading** slots of the formation constituents. This heading value is used in the determination of targets: those that lie along the line-of-sight of the formation are considered possible targets.

Cluster slot-values are computed by demons dedicated to the task, as shown by the procedural attachments above. Because demons are not restricted to pattern-matching functions, as are

many rule-based inference engines, extensive numerical computations can be included as part of rule-testing or execution. This ability to compute needed values "on the fly" means fewer facts need to be stored in a knowledge base. When seldom-used values are computed as they are needed, the knowledge base stays uncluttered and system efficiency increases.

Another example of numerical computation within a rule is the formation rule that gathers airborne objects into groups. Currently, criteria for membership in a formation is limited to physical proximity, but as knowledge about mission and formation tactics is added to the knowledge base, it can be used, too. As shown in the sample run in **Appendix B**, the **sayfrm-demon** that is a P.A. to the cluster frame's target slot prints a summary of formation information at the end of every time frame.

Knowledge Base

Besides containing the definitions of demons, the knowledge base, as it stands right now, contains definitions of enemy bases and friendly targets. Base information consists of a base name, location, and deployment percentage for each type of possible ID. We assume that pre-mission data might give us this type of information. As discussed in **Chapter 3**, plane deployment information at enemy bases is used in the cyclical

rule set to help determine ID, once combat radius and a list of possible originating bases is determined for each airborne object. Bases are "created" for a scenario using a **create-base** macro, which allows arbitrary positioning and deployment information to be entered. The x-y position is used in the calculation of an object's combat radius, to determine which bases it might have taken off from.

Target information consists of names, locations, and values of friendly sites, which might be strategic headquarters, troop concentrations, surface-to-air missile sites, or air-strips. This data is used in the determination of possible targets for formations of enemy planes. A site's position information is used in the calculation of the line-of-site to the enemy airborne object formation, which is then compared to the heading of the formation. If the difference is within a threshold of 10 degrees, the site is considered to be one of the formation's possible targets. A site's **value** parameter may be used later by a tactical situation assessment module that determines the size of the incoming formation based on the strength of airborne force needed to successfully attack the site.

Belief Maintenance

Although shown as a dotted box in **Figure 5**, a rudimentary belief-maintenance mechanism has been implemented for OAS2.

The dotted box is meant to indicate that any belief mechanism could fit into the modular design of OAS2, intercepting assertions from the demon "inference engine" on their way to the airborne object frames. OAS2's current belief mechanism, which uses a time-smoothing "filter" in the computation of ID probability, keeps a history of confidence assertions for each possible ID in a list. A "belief rating" is then computed for each ID by taking a time-weighted average of the current probability of ID vector and the assertions in the list. An interactive demonstration of the belief mechanism will clarify the operation.

First we create a demonstration airborne object called **demo-plane**. (The computers response is shown in *italics*; the ">" is the LISP prompt):

```
> (detect demo-plane)
Detected an airborne object...

(DEMO-PLANE)
```

Now we can **view** the demo-plane airborne-object frame, and examine its slots and values. Initially, each slot is empty.

```
> (view demo-plane)

F.568
    SELF ..... (F.568 () () () (#{Procedure 77}))
    CLASS ..... (AIRBORNE_OBJECT () () () ())
    PARENTS ..... ()
    NAME ..... (DEMO-PLANE () () () ())
    VELOCITY ..... (() () () () (#{Procedure 78}))
    XY-POS ..... (() () () (#{Procedure 79}) (#{Procedure
80}))
```

```
    HEADING ..... (() () () () ())
    RCS ..... (() () () () (#{Procedure 81}))
    IR_TEMP ..... (() () () () (#{Procedure 82}))
    RADAR ..... (() () () () (#{Procedure 83}))
    LRSIG ..... (() () () () (#{Procedure 84}))
    ENG ..... (() () () () (#{Procedure 85}))
    ENG_TYPE ..... (() () () () (#{Procedure 86}))
    LENGTH ..... (() () () () (#{Procedure 87}))
    WIDTH ..... (() () () () (#{Procedure 88}))
    MANEUVERS ..... (() () () () (#{Procedure 89}))
    ALTITUDE ..... (() () () (#{Procedure 90}) (#{Procedure
91}))
    PREV-ALTITUDE .... (() () () () ())
    HYP-PTR ..... (F.569 () () () ())
    COM_RAD ..... (() () () () (#{Procedure 92}))
    BASE_POO ..... (() () () () (#{Procedure 93}))
    POSS_IDS ..... (() () () () ())
    PTR_LIST ..... (() () () () ())
    PID_LIST ..... (() () () () ())
```

The frame above contains three slots that are used for belief-maintenance: the **POSS_IDS** slot contains a list of the possible identities of the airborne object; the **PTR_LIST** slot contains a list of pointers to child frames that more completely represent the ID classes in the **POSS_IDS** slot; and the **PID_LIST** slot contains a list of values that represent the probability of ID of each corresponding entry in the **POSS_IDS** slot. To see the function of these slots, we assert "ID is fighter, conf = .5":

```
> (id-update demo-plane 'fighter .5)
(1.0)
```

Now, if we **view** demo-plane, we see the effect of the assertion (only the belief-maintenance slots are shown in the remaining traces):

```
> (view demo-plane)
```

```
F.568
    NAME ..... (DEMO-PLANE () () () ())
    POSS_IDS ..... ((FIGHTER) () () () ())
    PTR_LIST ..... ((F.570) () () () ())
    PID_LIST ..... ((1.0) () () () ())
    PID%FIGHTER ..... ((0.5 0.0 0.0) () () () ())
```

The assertion that demo-plane's ID could be a fighter causes four things to happen: **fighter** is added to the list of possible IDs in the **POSS_IDS** slot; a subordinate fighter frame (F.570) is created to hold specific ID information about fighters, and a pointer to it is added to the **PTR_LIST** slot; a **PID%FIGHTER** slot is created to keep a history about all fighter assertions; and the **PID_LIST**, which keeps the current probability of ID vector, is updated.

In general, the **PID%obj-type**$_i$ slot keeps all past confidence values with which ID was asserted to be of type **obj-type**$_i$. A confidence rating, **CR**$_i$, is then computed for **obj-type**$_i$ based on the time-weighted average of the current probability value of **ID**$_i$ and the confidence values contained in the **PID%obj-type**$_i$ slot. The method used takes the current probability value and the last two asserted confidence values and weights them by .5, .3, and .1, respectively. (Because three values are used, when a new **PID%obj-type** slot is created, zeroes are inserted for the first two values.)

The final probability of ID vector is computed by taking **CR**$_i$ for each **obj-type**$_i$ and renormalizing so the **CR**$_i$s sum to

unity. This vector is then stored in the **PID_LIST** slot, with the first probability in the slot corresponding to the first ID listed in **POSS_IDS**, the second probability corresponding to the second ID, etc.

Of course, in the frame above, since there is only one possible ID so far, the **PID_LIST** value is 1.0. However, when we assert "ID is bomber, conf = .3", we'll see the belief values distributed between bomber and fighter.

```
> (id-update demo-plane 'bomber .3)
(0.625 0.375)

> (view demo-plane)

F.568
     NAME ............. (DEMO-PLANE () () () ())
     POSS_IDS ......... ((FIGHTER BOMBER) () () () ())
     PTR_LIST ......... ((F.570 F.571) () () () ())
     PID_LIST ......... ((0.625 0.375) () () () ())
     PID%FIGHTER ...... ((0.5 0.0 0.0) () () () ())
     PID%BOMBER ....... ((0.3 0.0 0.0) () () () ())
```

We see now that the .625 to .375 ratio in the **PID_LIST** slot is equal to the initial values in the **PID%FIGHTER** and **PID%BOMBER** slots. (When the prior probability equals 1.0, it is not used in the time-weighting scheme.)

If we want to assert ID information about *specific* fighter or bomber types, like MiG-29 or Tu-95, we need a place to store this information. That is the purpose of the fighter and bomber frames pointed to by the elements of the **PTR_LIST** slot.

If we view F.570 and F.571, the frames representing specific fighters and bombers, we see that they are both currently empty.

```
> (view 'f.570)

F.570
    SELF ..... (F.570 () () () ())
    CLASS ..... (FIGHTER () () () ())
    PARENTS ..... (F.568 () () () ())
    POSS_IDS ..... (() () () () ())
    PID_LIST ..... (() () () () ())
> (view 'f.571)

F.571
    SELF ..... (F.571 () () () ())
    CLASS ..... (BOMBER () () () ())
    PARENTS ..... (F.568 () () () ())
    POSS_IDS ..... (() () () () ())
    PID_LIST ..... (() () () () ())
```

However, they come into play when we assert "ID is MiG-29, conf = .7". This assertion causes MiG-29 to be added to the **POSS_IDS** slot of the fighter frame, F.570, where belief-maintenance occurs for ID assertions about specific types of fighters. In addition, the assertion has an effect on the parent frame, demo-plane, since MiG-29 is of class fighter. It is as if the assertion "ID is *fighter*, conf = 0.7" were made.

```
> (id-update demo-plane 'MiG-29 .7)
(1.0)

> (view demo-plane)

F.568
    NAME ..... (DEMO-PLANE () () () ())
    POSS_IDS ..... ((FIGHTER BOMBER) () () () ())
    PTR_LIST ..... ((F.570 F.571) () () () ())
    PID_LIST ..... ((0.692 0.308) () () () ())
    PID%FIGHTER ..... ((0.7 0.5 0.0 0.0) () () () ())
    PID%BOMBER ..... ((0.3 0.0 0.0) () () () ())
```

The confidence value 0.7 has been added to the **PID%FIGHTER** slot and **PID_LIST** has been updated.[58] In addition, the child fighter-frame to demo-plane has been updated.

```
> (view 'f.570)

F.570
     SELF ..... (F.570 () () () ())
     CLASS ..... (FIGHTER () () () ())
     PARENTS ..... (F.568 () () () ())
     POSS_IDS ..... ((MIG-29) () () () ())
     PID_LIST ..... ((1.0) () () () ())
     PID%MIG-29 ..... ((0.7 0.0 0.0) () () () ())
```

MiG-29 was added to the **POSS_ IDS** slot of F.570, because it was not a member already, the **PID_LIST** slot was set to 1.0, and the **PID%MIG-29** slot was created with an initial value of 0.7, which means, "Given that ID is fighter, Prob(ID is MiG-29) = 0.7". If we now assert another specific ID of type fighter, the proba- bilities will be distributed between IDs just as in **demo-plane**.

```
> (id-update demo-plane 'Su-27 .85)
(0.452 0.548)
```

---

58. The ratio .692/.308 is equal to the ratio $CR_{fighter}/CR_{bomber}$, where $CR_{fighter}$ = [(.5 * .625) + (.3 * .7) + (.2 * .5)] and CR_{bomber} = [(.5 * .375) + (.3 * .3) + (.2 * 0.0)]. The values .5, .3, and .2 are weighting factors, and .625 and .375 are the prior probabilities of ID.

```
> (view demo-plane)

F.568
    NAME ..... (DEMO-PLANE () () () ())
    POSS_IDS ..... ((FIGHTER BOMBER) () () () ())
    PTR_LIST ..... ((F.570 F.571) () () () ())
    PID_LIST ..... ((0.752 0.248) () () () ())
    PID%FIGHTER ..... ((0.85 0.7 0.5 0.0 0.0) () () () ())
    PID%BOMBER ..... ((0.3 0.0 0.0) () () () ())
```

The confidence value 0.85 has been added to the **PID%FIGHTER** slot, since an Su-27 is of class fighter. When we **view** the child fighter frame:

```
> (view 'f.570)

F.570
    SELF ..... (F.570 () () () ())
    CLASS ..... (FIGHTER () () () ())
    PARENTS ..... (F.568 () () () ())
    POSS_IDS ..... ((MIG-29 SU-27) () () () ())
    PID_LIST ..... ((0.452 0.548) () () () ())
    PID%MIG-29 ..... ((0.7 0.0 0.0) () () () ())
    PID%SU-27 ..... ((0.85 0.0 0.0) () () () ())
```

We see that Su-27 has become one of the possible IDs, and the **PID_LIST** has been adjusted accordingly. Another assertion about Su-27:

```
>  (id-update demo-plane 'Su-27 .95)
(0.374 0.626)

> (view demo-plane)

F.568
    NAME ..... (DEMO-PLANE () () () ())
    POSS_IDS ..... ((FIGHTER BOMBER) () () () ())
    PTR_LIST ..... ((F.570 F.571) () () () ())
    PID_LIST ..... ((0.795 0.205) () () () ())
    PID%FIGHTER ..... ((0.95 0.85  0.7  0.5  0.0 0.0) () () ()
())
    PID%BOMBER ..... ((0.3 0.0 0.0) () () () ())
```

The 0.95 assertion on Su-27 has been added to **demo-plane**, increasing the likelihood of Prob(ID is fighter) to 0.795.

```
> (view 'f.570)

F.570
    SELF ..... (F.570 () () () ())
    CLASS ..... (FIGHTER () () () ())
    PARENTS ..... (F.568 () () () ())
    POSS_IDS ..... ((MIG-29 SU-27) () () () ())
    PID_LIST ..... ((0.375 0.625) () () () ())
    PID%MIG-29 ..... ((0.7 0.0 0.0) () () () ())
    PID%SU-27 ..... ((0.95 0.85 0.0 0.0) () () () ())
```

In addition, Prob(ID is Su-27 | ID is fighter) is now 0.625. If we assert that ID is bomber with a very low confidence, it increases the probability that ID is fighter:

```
> (id-update demo-plane 'bomber .1)
(0.816 0.184)

> (view demo-plane)

F.568
    NAME ..... (DEMO-PLANE () () () ())
    POSS_IDS ..... ((FIGHTER BOMBER) () () () ())
    PTR_LIST ..... ((F.570 F.571) () () () ())
    PID_LIST ..... ((0.816 0.184) () () () ())
    PID%FIGHTER ..... ((0.95 0.85 0.7 0.5 0.0 0.0) () () ()
())
    PID%BOMBER ..... ((0.1 0.3 0.0 0.0) () () () ())
```

This concludes the demonstration of the belief mechanism. Part of the reason that the current value of Prob(ID) is included in the computation of the new value of Prob(ID) is to "smooth" the transition in probability in case of a "quirk" sensor return that incorrectly asserts ID. With this method, several assertions upon ID_i must be made for Prob(ID_i) to be

significantly shifted towards that ID_i. It should also be noted that low-confidence assertions upon ID increase all the other ID's probability, as shown by the above assertion, "ID is bomber, conf = 0.1".

Overall, this rather ad-hoc time-smoothing belief mechanism has little sophistication, especially when it comes to combining multiple evidence upon a single hypothesis. The methodology, however, is conceptually valid in at least one respect: the most recently received sensor data is assigned the most validity. This is important because sensors may provide inaccurate information at long range, and later correct their prior assertions when the airborne objects get closer. On the other hand, a sensor may provide *more* accurate information at long range if jamming or poor weather conditions at close range cause degradation of sensor performance. Perhaps a future implementation of the expert system can decide when to disregard sensor returns, even if they are the latest to have been received.

For future work, we recommend the Shafer representation and Dempster's rule of combination to provide an inferencing technique for sensor evidencial reasoning. Several previous efforts in the field[59] are serving as the basis for development

---

59. See **References**.

work. We expect that Shafer-Dempster theory will provide greater accuracy and efficiency than the Bayesian inference net of OAS1. The former is purported to provide greater inferencing capability when the number of sensors and IDs is large. This is because the use of standard Bayes rule inferencing requires the knowledge of appropriate statistical data for the ID and sensor relationships, which is often unavailable as system complexity increases.[60] The efficiency using Shafer-Dempster should also increase, since ID probability updates need only be done when new sensor evidence is received.

A Sample Run

OAS2's operation is depicted in a sample run in **Appendix B**. Shown is the input trackfile that drives the system, the response of the system, an internal trace of demon actions, and the knowledge representation of frame objects. Full-length airborne object frames are listed, and a trace is shown of the demon driver code that selectively spawns, executes, and kills demons. Criteria for cyclical rule invocation, requests for sensor information, and future improvements to the system are also discussed.

---

60. In addition, it is difficult to specify consistent degrees of belief with Bayesian inferencing, as shown by [Shafer, 1976, p.24]. Also see the introduction of [Lowrance and Garvey, 1983].

Summary of OAS2: Strengths

OAS2 has satisfied its design goals by reducing dependency on a pre-packaged expert system tool (ARF), resulting in a modular system that is flexible and extensible. This, in turn, has increased system efficiency: OAS2's sample scenarios process three times the number of airborne objects as OAS1 in one-sixth the time--a factor of nearly twenty in increased performance. Running the scenario of **Appendix B**, the non-optimized[61] system with compiled code runs nearly in real-time, with the use of 30-second time frames. In addition, the system uses only one-ninth the heap space[62] of OAS1 for the same number of objects, resulting in much less frequent garbage-collections.[63]

System extensibility is enhanced by the use of demons to implement inferencing capability and the use of individual frames to hold information about airborne objects. As previously mentioned, any number of demons can be defined in the system without seriously affecting performance. This means we can add rules at any time by coding a few lines of LISP. In addition, since demons can execute general procedures, rather

---

61. Steps have not been taken to reduce the execution time of the system by exploiting redundancy; it is estimated we could increase performance by 50% if we optimized the code.

62. Heap space refers to storage area in a LISP environment.

63. Garbage-collection is a computer operation in a LISP environment that reclaims unused memory cells.

that being restricted to pattern-matching or testing if-then constructs, we have the freedom to code rules that print messages, organize formations, or wait for the occurrence of events.

Efficiency is further enhanced by having procedural attachments spawn only those demons that have a chance at executing, and by using meta-demons to eliminate unnecessary demon testing. Before, in OAS1, each rule was checked on every detected airborne object. Now, only selected demons need be checked upon a limited number of airborne objects.

Other advantages of OAS2 include its simple operation and clean interface between modules. All system tasks performed upon the reception of new sensor evidence are initiated by a single entry in the input trackfile. From there, returns are stored in the knowledge base, and procedural attachments send appropriate demons to the agenda, where they wait to be tested and executed. Tests involve accessing the knowledge base (or asking for information when it's unavailable), while executions spawn other demons, perform belief-maintenance, or make assertions to the knowledge base.

The demon-oriented architecture of OAS2 creates a "distributed" inference engine, in that spawnings, killings, and executions of demons are relatively independent processes,

and, to a certain extent, can take place in any order. This makes OAS2 an especially good candidate for implementation in a parallel-processing environment. One scheme would be to have unused processors take the next demon waiting in the queue and process it until completion. If this were accompanied by a mechanism that could accept sensor returns asynchronously, rather that in sequence, the performance of the system would be further improved.

The modular design of OAS2 will make it easy to add a belief-mechanism to the system. Shafer-Dempster or Bayesian-based belief trees can be associated with each airborne object frame, as is done with the current time-smoothing belief algorithm. An additional benefit of the Shafer-Dempster implementation is that updates of the tree need only be performed when knowledge of ID is necessary, rather than each time new evidence is received.

Because we have reduced system dependency on ARF, almost any frame-based knowledge representation system can now be used in OAS2, if it has some provision for procedural attachments. Use of a more streamlined knowledge representation mechanism than ARF would likely improve the system's performance, since the overhead for rules, STMs, and other frame objects that support the pattern-matching inference engine in ARF would not be necessary. Another development effort, in fact, has

proceeded in parallel with OAS2, resulting in a frame system with P.A.s that has four times the execution speed of ARF. This task was performed by Craig Lee of Hughes [Lee, 1985].

Finally, OAS2 provides some of the features listed in **Chapter 3** as characteristic of an ideal object analysis expert system. These include the ability to provide requests for sensor information to a sensor manager, and the system structure to provide mission-level inferencing about formations, targets, and intent of enemy planes.

Weaknesses of OAS2

OAS2 has no serious structural problems, and most of the shortcomings of the system can be overcome by further development work. Unlike OAS1, OAS2 provides an extensible system structure, so the elimination of problems should not be too difficult.

A current weakness of OAS2 is its unsophisticated I/O facility: it is difficult both for the user to easily develop new input scenarios and to see the result of system inferences. New input trackfiles are relatively time-consuming to develop because a sensor return for each sensor must be specified for each airborne object for every time frame of the scenario. To thoroughly evaluate the expert system, a variety of scenarios need to be developed. Examination of performance in different

situations helps the system designer determine the confidence values of rules, discover the relative importance of different sensor returns, and prioritize sensor use.

The iterative nature of expert system development suggests that new scenarios should be easy to develop. Ideally, for a scenario like that of **Appendix B**, a user should simply have to specify the IDs of the six incoming aircraft, their mission, and their flight path. From this data, a preprocessor would then create an entire input trackfile that could be loaded into OAS2. Development on a "sensor simulator", in fact, has already begun; it is discussed more fully in the next chapter.

Turning to the system output of OAS2, the current implementation has limited capability. While OAS2 does identify the detection of airborne objects and print out formation and target information, there is no way to explain the decision process of the system or trace the current Prob(ID) of airborne objects.

The requirement for an explanation facility may be made easier because rules are implemented as demons. The existing demon-tracing facility[64] provides a skeleton for a more advanced explanation facility, which would allow a user to

---

64. See the demon traces in **Appendix B**.

determine the inference process through which a piece of knowledge was discovered, as well as record the frequency with which rules were used. The relative importance of rules could then be investigated in anticipation of the time when the ratio {rule importance/rule cost} determines how to allocate scarce sensor resources.

The problem of presenting the user with the current Prob(ID)s for all objects detected could be solved by installing a query facility into the system, or by presenting the user with periodic updates on ID upon the reception of new evidence. The first solution is probably what will eventually be implemented in OAS2, since the use of Shafer-Dempster theory for belief-maintenance allows Prob(ID) to be updated whenever it is needed. The explanation facility would contain a simple natural language interface, allowing the user to ask about mission, phase, and ID of various craft he opposes. In addition, there will very likely be a graphics display connected to the output of the system, as in OAS1. Much of the required information could then be presented in tabular and graphic form, reducing the frequency of user requests.

CHAPTER 6
SUMMARY OF RESULTS
AND RECOMMENDATIONS FOR FUTURE RESEARCH

Introduction

This final chapter discusses some of the major findings of this research, summarizes important aspects of the two versions of the object analysis expert system, and presents recommendations for future extensions to OAS2.

OAS1 Summary

The first cut at the expert system, OAS1, served primarily as a tool to learn about requirements for the object analysis task. OAS1 demonstrated the feasibility of performing object analysis with an expert system, and with the graphics front-end, helped to demonstrate the impact of artificial intelligence techniques on avionics. In addition, OAS1 served as a vehicle to evaluate ARF, explore the requirements of object analysis belief maintenance, and identify important rules and knowledge areas.

OAS1 performed object analysis reasonably well, but suffered in power, flexibility, and extensibility because it was built on top of an integrated inference engine/rule base/confidence propagation package. With OAS1's emphasis on frames to represent system objects, we were unable to implement all the rules we gathered through knowledge engineering.

For a prototype expert system, we really wanted to be able to investigate different knowledge representation schemes and rule formats. Furthermore, we wanted to be able to evaluate different confidence propagation interfaces with the belief maintenance mechanism.

A positive result of OAS1 was the Bayesian inference net. Though the ID-update process is time-consuming, it maintained belief well, and was able to handle incorrect evidencial assertions without producing sudden "jumps" in probability of ID. The Bayesian net accurately describes relationships between sensor evidence and identity through probability matrices, and represents evidence in a tree structure that reflects the ordering of information in the problem domain.

Of the rules that were implemented, we found that the cyclical rule set relating ID to combat radius, then to point-of-origin and deployment data, and finally back to identity, worked very well. Correct ID was achieved more quickly using the cyclical rules, with eighteen percent fewer sensor assertions required.

Finally, the graphics interface we added to OAS1 proved to be very useful. Besides its value as a marketing tool, the overlay helped to identify useful methods of representing information to the eventual end-user, the pilot. We realized

that the pilot needs to be informed of enemy IDs and formations, but that he should also have some way to query the system for additional information. Finally, we identified the eventual need for an explanation mechanism to justify inferences to the pilot.

OAS2 Summary

With the shortcomings of OAS1 in mind, we designed OAS2 to have a much more modular structure. We implemented rules as demons, using their built-in control structure as an inference engine to perform spawn, test, execute, and kill functions. OAS2 has a clean interface between evidencial input (sensor returns) and the inference engine, accomplished through the use of slot procedural attachments to spawn demons and meta-demons. This interface promotes system efficiency: only those rules that have a chance of firing are ever tested. Furthermore, the number of defined demons does not affect the execution time of the system. In contrast, OAS1's execution time increases linearly as a function of the number of *defined* rules, since each rule in the rule base is examined when the inference engine is invoked.

Demons' ability to execute general code, coupled with the centralization in frames of knowledge about planes and formations, allows implementation of all the rules mentioned in Chapter 3. OAS2 performs high-level inferencing about airborne

object formations, missions, and mission-phases, and has the structure necessary for extension to the tactical situation assessment task. OAS2 also provides requests for missing sensor returns, and hypothesizes possible targets for plane formations.

One design goal of OAS2 was to make the system modular enough so that different belief maintenance mechanisms could be evaluated with a working expert system.[65] Though we did not develop a sophisticated belief system for OAS2, we came up with an ad-hoc "time-smoothing" mechanism. Though unable to combine multiple assertions, it has a valid methodology of assigning the most confidence to recently-received sensor data. By using a rudimentary belief mechanism in OAS2, we demonstrated the feasibility of keeping belief information *with* frame objects, whether those frames represent classes (fighter) or specific types (MiG-29) of airborne objects.

Recommendations For Future Research

OAS2 provides a promising system design upon which to build an operational system. To become truly useful, however, much more knowledge must be embodied in the system. Research must be done in the following areas: formation-level rules (what

---

65. One of the next plans for OAS2 is the integration of a Shafer-Dempster belief mechanism, the development of which took place in parallel to that of OAS2.

formation constituencies and actions help to determine ID); mission and phase rules (how does mission affect ID, what does one plane's mission imply about other planes in the same formation, how does current phase determine ID and future maneuvers); and rules to distinguish between planes of the same class (what distinguishes a MiG-29 fighter from an Su-27 fighter, what characterizes two MiG-29s outfitted for different missions). Other sources of higher-level knowledge will become available when Defense Systems Corporation completes its pilot interviews, and when outside sources, such as JTIDS and AWACS become integrated into the system.

Additional research must also be performed on lower-level sensor attributes, to find out how they determine parameters currently in use in OAS2. For example, OAS2 now assumes that RWR sensors give the specific radar type of airborne objects. In reality, however, RWRs give pulse repetition frequency, signal coding, radio frequency, and other low-level attributes that together determine radar type. So the expert system will more accurately reflect real-world sensor inputs, additional rules need to be developed and installed in OAS2.

One feature of OAS1 that is lacking in OAS2 is a confidence propagation mechanism. This allows both sensor returns and rules to have associated "belief factors", which are combined according to some algorithm to yield resultant assertions that

also have associated confidence measures. Various algorithms must be evaluated, and an interface to the belief mechanism chosen.

Lack of input/output facilities are a particular weakness of OAS2. Currently, system input consists of sequence of sensor data that must be hand-coded for each different scenario. Because the performance characteristics of sensors are generally "fixed", a better solution is to develop a simulator that embeds sensor characteristics in software.

For example, to simulate a radar sensor, we would gather its performance characteristics: detection range, angular resolution, jamming and weather sensitivity, base frequency, scan rate, etc. In addition, since some capabilities vary as a function of the type of plane being examined, we would have to encode these also. Once a sensor is "defined" in this manner, the user would need only to specify the airborne objects in a scenario, and the simulator would output the sequence of returns the sensor would provide.

Use of a sensor simulator has many benefits:

(1) the user no longer has the cumbersome task of hand-coding trackfiles

(2) the rate at which sensor returns are received can be easily varied

(3) a sensor can be completely "turned off", if desired, to evaluate expert system performance without it

(4) new airborne object types can be added to the expert system by simply augmenting current sensor performance characteristics

(5) a simulated sensor manager could be developed, "closing the loop" by integrating expert system "requests for information" into the output of the sensor simulator

The simulator is current being designed by Craig Lee and Mark Young of Hughes. In some ways, depending on how extensively development proceeds, the sensor simulator resembles another expert system.

The output facilities of OAS2 need to be improved. The graphics overlay of OAS1 provides some insight into how to effectively present information to a pilot, buy additional graphical formats need to be developed and shown to pilots for evaluation. The feedback they give may help prioritize our presentation of information, which may, in turn, affect our methods of knowledge acquisition and representation.

Along the lines of user interface, an explanation facility needs to be developed. As previously mentioned, this may be constructed on top of the demon tracing facility already present in OAS2. As the system's rule and knowledge base grows, we will increasingly need a mechanism for tracing and justifying knowledge inference through rules.

Finally, we need to develop methods of evaluating the performance of the expert system, to provide justification for its inclusion in an avionics suite. Possible evaluation criteria include time-to-ID (how much sooner can we identify an airborne object with the expert system than by using a human operator), launch-zone improvement (how much larger is the range at which ownship can fire air-to-air missiles at incoming enemy objects), and resources used per target (how many sensors do we need and how long must they "look" at enemy objects to accurately identify them).

APPENDIX A

This appendix shows a sample run of OAS1, the first-cut at the object analysis expert system, along with some sample frames that depict the ARF representation of rule sets and airborne object patterns. In addition, two sample screens of the graphic output of the system are shown. A description of the scenario represented by these screens is included prior to the sample trace.

Rule Set Frames

Shown here are the frame representations for the rule sets used in OAS1. The eighteen rules implemented have been divided into four groups: rule set 1 relates velocity and radar cross-section rules to ID, rule set 2 relates IR temperature to acceleration facts, rule set 3 relates rate-of-climb (ROC) to acceleration and ID, and rule set 4 is the cyclical rule set described in Chapter 3. As code is loaded into the LISP environment, rules are converted into ARF frames, which also represent STMs, FACTs, and user-defined patterns like airborne-objects. (Some of these other frame representations are shown at the end of this section.) The system responds with:

```
Creating Rule Set - F.36 - ID-PLANE-1
Creating Rule Set - F.45 - ID-PLANE-2
Creating Rule Set - F.54 - ID-PLANE-3
Creating Rule Set - F.59 - ID-PLANE-4
```

These four messages indicate that four RULE frames have been instantiated. ID-PLANE-1 corresponds to rule set 1, which is stored in frame object F.36. Once created we can view the frames (the ">" is the LISP prompt):

```
> (view 'f.36)

F.36
    SELF ..... (F.36 () () () (#{Procedure 66}))[66]
    CLASS ..... (RULE-SET () () () ())
    PARENTS ..... ()
    DOC ..... ("Rule set 1: track-speed facts" () () () ())
    NAME ..... (ID-PLANE-1 () () () ())
    FIRE ..... (SEQUENTIAL () () () ())
    DO ..... (ALL () () () ())
    LOCALS ..... (() () () () ())
    RULES ..... (($RULE-CHECK-SPEED-1 $RULE-CHECK-SPEED-2
$RULE-CHECK-SPEED-3 $RULE-CHECK-SPEED-4) () () () ())
    CODE ..... (RULE-SET-ID-PLANE-1 () () () ())
    TRACE ..... (() () () () ())
```

This is a sample of ARF's representation for a frame. Frames have slots (such as SELF, CLASS, and PARENTS), and slots have values (like F.36, RULE-SET, and ()). Except for the PARENTS slot, each slot in ARF has five positions: the first position holds the value of the slot, and the remaining four contain **procedural attachments**, LISP code that can be executed upon accesses of the slots. (This last feature is not used explicitly in OAS1, but is used extensively in OAS2.) The **F.36** frame object is the parent frame for each of the four rules that constitute the rule set with **NAME** ID-PLANE-1. Each of the four rules is itself represented by a frame, pointers to which are contained in the **RULES** slot of F.36. The parent frames for the other three rule sets (ID-PLANE2, ID-PLANE3, and ID-PLANE4) are shown below:

---

66. #{Procedure x} is ARF's representation for a procedural attachment.

```
> (view 'f.45)

F.45

    SELF ..... (F.45 () () () (#{Procedure 66}))
    CLASS ..... (RULE-SET () () () ())
    PARENTS ..... ()
    DOC ..... ("Rule set 2: infer accel.ation facts" () () () ())
    NAME ..... (ID-PLANE-2 () () () ())
    FIRE ..... (SEQUENTIAL () () () ())
    DO ..... (ALL () () () ())
    LOCALS ..... (() () () () ())
    RULES ..... (($RULE-CHECK-ACCELERATION $RULE-CHECK-DELTA-IR-1
$RULE-CHECK-DELTA-IR-2 $RULE-CHECK-DELTA-IR-3) () () () ())
    CODE ..... (#{Syntax 50 RULE-SET-ID-PLANE-2} () () () ())
    TRACE ..... (() () () () ())

> (view 'f.54)

F.54
    SELF ..... (F.54 () () () (#{Procedure 66}))
    CLASS ..... (RULE-SET () () () ())
    PARENTS ..... ()
    DOC ..... ("Rule set 3: infer ROC facts" () () () ())
    NAME ..... (ID-PLANE-3 () () () ())
    FIRE ..... (SEQUENTIAL () () () ())
    DO ..... (ALL () () () ())
    LOCALS ..... (() () () () ())
    RULES ..... (($RULE-CHECK-DELTA-ROC-1 $RULE-CHECK-DELTA-ROC-2)
() () () ())
    CODE ..... (RULE-SET-ID-PLANE-3 () () () ())
    TRACE ..... (() () () () ())

> (view 'f.59)

F.59
    SELF ..... (F.59 () () () (#{Procedure 66}))
    CLASS ..... (RULE-SET () () () ())
    PARENTS ..... ()
    DOC ..... ("Rule set 4: base, comrad, ID cyclical facts"
() () () ())
    NAME ..... (ID-PLANE-4 () () () ())
    FIRE ..... (SEQUENTIAL () () () ())
    DO ..... (ALL () () () ())
    LOCALS ..... (() () () () ())
    RULES ..... (($RULE-CHECK-ID-COMRAD-1 $RULE-CHECK-ID-COMRAD-2
$RULE-CHECK-ID-COMRAD-3 $RULE-CHECK-COMRAD-BASE-1 $RULE-CHECK-COMRAD-BASE-2
$RULE-CHECK-COMRAD-BASE-3 $RULE-CHECK-BASE-ID-1 $RULE-CHECK-BASE-ID-2)
() () () ())
    CODE ..... (RULE-SET-ID-PLANE-4 () () () ())
    TRACE ..... (() () () () ())
```

A Sample Run of OAS1

Below is a sample run of OAS1. The input sequence of sensor returns (trackfile) that drives the simulation is interlaced with OAS1's output, so the effect of each newly-asserted fact can be seen. Trackfile FACTS and DETECTIONS are shown justified at the left; system outputs are shown indented. As mentioned before, sensor returns have been deliberately separated into facts that go to the inference net and facts that go to an STM; these are distinguished by the two functions that assert sensor returns, ***fact*** and **assert-data**. The separation was done only for efficiency purposes. In an operational expert system, some sort of preprocessor would decide where to send the different sensor returns.

One other efficiency tactic that would be eliminated in an operational system was to *give* the expert system **delta-quantities**, rather than have it compute them itself. For instance, rather than give the expert system IR temperature in each time frame and have it keep track of temperature changes over time, we give it delta-IR temperature directly. As has been stressed before, the main purpose of OAS1 was to determine the rough structure of the object analysis task, and not to worry about details.

The scenario represented by this simulation is ownship facing two incoming enemy fighters, one MiG-29 and one Su-27. The sensor data at the beginning of the run has been purposely "skewed" to make the expert system "believe" that the two incoming objects are actually bombers or transports. This was done to see how the expert system would recover from erroneous sensor data.

Time for the simulation has been broken into seven, thirty-second frames, corresponding to "bursts" of sensor data that might be received. (Future systems that can process returns closer to real-time might accept data asynchronously.)

As discussed in Chapter 4, the primary output of OAS1 is a six-element vector corresponding to probabilities of the six possible airborne object identities we have allowed: (Tu-95 Tu-144 MiG-29 Su-27 AA-9 AA-7). The Tu-95 is a bomber, the Tu-144 a transport, MiG-29 and Su-27 are fighters, and AA-9 and AA-7 rockets. If the system works perfectly, at the end of the run the ID vector for PLANE0 (the first detected airborne object) will be equal to (0.0 0.0 1.0 0.0 0.0 0.0), indicating ID is MiG-29, and the ID vector for PLANE1 (the second detected airborne object) will be equal to (0.0 0.0 0.0 1.0 0.0 0.0), indicating ID is Su-27.

All OAS1 system responses (except for status messages) shown indented below are in one of two formats:

(1) {AO frame} {sensor attribute} : {sensor value} = {confidence level}
P(ID) #{AO number}: (P_1 P_2 P_3 P_4 P_5 P_6),

or

(2) {AO frame} {rule number} : {assertion vector}
P(ID) #{AO number}: (P_1 P_2 P_3 P_4 P_5 P_6),

where P_1 = Prob (ID is Tu=95)
P_2 = Prob (ID is Tu-144)
P_3 = Prob (ID is MiG-29)
P_4 = Prob (ID is Su-27)
P_5 = Prob (ID is AA-9)
P_6 = Prob (ID is AA-7) ,

everything not enclosed in curly-braces {} is literal punctuation, and AO stands for airborne object.

Output (1) is a system response to a FACT assertion (by *fact*), and output (2) is a system response when a rule fires. {sensor attribute} in (1) is one of the eight nodes in **Figure 4**, and {sensor value} is one of the possible values of that node. The operation of an {assertion vector} in (2) is discussed on page 61 in Chapter 4.

OAS1 uses eighteen rules, and the numbers used in (2) correspond to rules as follows:

R0 thru R3	velocity and radar cross-section rules
R4 thru R6	infrared and acceleration rules
R7 thru R13	rate-of-climb, engine, and IR rules
R14 thru R17	cyclical rule set

The meanings of the two types of system outputs should become clear upon examination of the trace below. Explanatory comments are shown to the right in *italics*.

```
(time 0)                                          Trackfile e try

    time = 0                                      System response to that entry

    Object Analysis Expert System Demo

(plane-detect)                                    Radar has detected a plane

    [Binding PLANE0]                              call it PLANE0
    Creating STM - F.157 - Basic blackboard       an STM to keep FACTS on PLANE0
                         - list of objects.

(*fact*  plane0   'track-speed   .90    1.0)      assert to the STM that PLANE0's
                                                  velocity = Mach  0.9, with conf
= 1.0

(assert-data  plane0  LRSIG     SU2     .45)      assert to PLANE0's inf net
                                                  that Long-Range signature is
                                                  Soviet-type 2, conf = 0.45

    PLANE 0 lr_sig :  su2=.450                    Prob(ID is Tu-95) = .201
    P(ID) #0: (.201 .201 .138 .197 .130 .130)     Prob(ID is Tu-144) = .201

(assert-data  plane0  IR-exh   300      .28)      IR temp < 300 deg,
                                                  conf = .28

    PLANE 0 ir_ex_t:  <300=.280                   Asserted to net...
    P(ID) #0: (.216 .215 .131 .188 .124 .124)     Current Prob(ID) vector

                                                  End of frame, so rule set
                                                  begins to execute:

    PLANE 0 R0: .237 .237 .237 .237 .025 .025     Rule 0 => ID not a rocket
    P(ID) #0: (.278 .276 .169 .242 .016 .016)     since low vals in AA-9 and
                                                  AA-7 assertion slots

                                                  Only one rule fired

(time 30)

    time = 30                                     Next time frame

(plane-detect)                    ; plane # 1     Second plane detected, and
                                                  associated STM and belief trees
    [Binding PLANE1]                              are created
    Creating STM - F.163 - Basic blackboard
                         - list of objects.
```

```
(*fact*  plane0  'track-speed   1.30     1.0)
(*fact*  plane1  'track-speed   1.20     1.0)

(*fact*  plane0  'RCS           'medium  .50)
(*fact*  plane1  'RCS           'medium  .65)

(assert-data  plane0  LRSIG     SU2      .65)

    PLANE 0 lr_sig :  su2=.650
    P(ID) #0: (.298 .297 .125 .256 .011 .011)

(assert-data  plane1  LRSIG     SU1      .35)

    PLANE 1 lr_sig :  su1=.350
    P(ID) #1: (.246 .271 .222 .235 .012 .012)

(assert-data  plane0  ENG       quad     .30)

    PLANE 0 #eng   :  eng4=.300
    P(ID) #0: (.328 .318 .109 .223 .009 .009)

(assert-data  plane1  ENG       quad     .28)

    PLANE 1 #eng   :  eng4=.280
    P(ID) #1: (.264 .286 .207 .219 .011 .011)

(assert-data  plane0  IR-exh    300      .33)

    PLANE 0 ir_ex_t:  <300=.330
    P(ID) #0: (.350 .337 .096 .197 .008 .008)

(assert-data  plane1  IR-exh    300      .35)

    PLANE 1 ir_ex_t:  <300=.350
    P(ID) #1: (.296 .315 .178 .189 .009 .009)

(assert-data  plane0  Radar-sig unknown  .50)

    PLANE 0 r-sig  :  r0=.500
    P(ID) #0: (.333 .321 .091 .187 .033 .033)

(assert-data  plane1  Radar-sig RDR0     .25)

    PLANE 1 r-sig  :  r0=.250
    P(ID) #1: (.294 .313 .177 .188 .012 .012)

    PLANE 0 R0: .237 .237 .237 .237 .025 .025
    P(ID) #0: (.333 .321 .091 .187 .033 .033)

    PLANE 1 R0: .237 .237 .237 .237 .025 .025
    P(ID) #1: (.294 .313 .177 .188 .012 .012)
```

To STMs: velocity info

To STMs: radar cross-section info

To net: new LRSIG

ID still looks like bomber or transport

ID vector isnt the same as that of PLANE0, because fewer sensor assertions received

Number of eng = 4, conf = .3

Looks even more like large, four-engined plane

Same for PLANE1

Still low temp

Detected radar, but unknown

Rule0 fires again with the same assertion vector: note how it doesn't change the Prob(ID) vector at all

```
(time 60)

    time = 60                                              New time frame

(*fact*  plane0  'delta-ROC      6        .95)    Change in alt. = 6000 feet
(*fact*  plane1  'delta-ROC      7        .95)

(*fact*  plane0  'track-speed   1.85      .95)    velocity has increased sharply
(*fact*  plane1  'track-speed   1.85      .95)

(assert-data  plane0  LRSIG     SU2       .75)

    PLANE 0 lr_sig :  su2=.750
    P(ID) #0: (.347 .334 .074 .193 .024 .024)

(assert-data  plane1  LRSIG     SU1       .50)

    PLANE 1 lr_sig :  su1=.500
    P(ID) #1: (.292 .323 .176 .191 .007 .007)

(assert-data  plane0  ENG       quad      .55)    Still see "wrong" number of eng

    PLANE 0 #eng   :  eng4=.550
    P(ID) #0: (.444 .404 .035 .092 .011 .011)    Really think ID is bomber or
                                                  transport for PLANE0 and 1
(assert-data  plane1  ENG       quad      .60)

    PLANE 1 #eng   :  eng4=.600
    P(ID) #1: (.417 .429 .070 .076 .003 .003)

(assert-data  plane0  IR-exh   1000       .55)    Getting hotter: not a charac-
                                                  teristic of a bomber

    PLANE 0 ir_ex_t:  <1000=.550
    P(ID) #0: (.368 .369 .065 .177 .009 .009)

(assert-data  plane1  IR-exh   1000       .35)

    PLANE 1 ir_ex_t:  <1000=.350
    P(ID) #1: (.363 .398 .111 .119 .003 .003)

(assert-data  plane0  Radar-sig Foxfire   .20)    Now get radar like MiG-29

    PLANE 0 r-sig  :  foxf=.200
    P(ID) #0: (.360 .362 .076 .198 .002 .002)

(assert-data  plane1  Radar-sig Hilark    .25)    Get radar like Su-27

    PLANE 1 r-sig  :  hi-l=.250
    P(ID) #1: (.350 .384 .107 .152 .002 .002)
```

```
    PLANE 0 R2: .114 .114 .271 .271 .114 .114
    P(ID) #0: (.278 .279 .118 .320 .001 .001)

    PLANE 1 R2: .099 .099 .300 .300 .099 .099
    P(ID) #1: (.230 .252 .212 .301 .001 .001)

    PLANE 0 R8: .137 .137 .451 .451 .137 .137
    P(ID) #0: (.138 .139 .194 .525 .000 .000)

    PLANE 1 R8: .137 .137 .451 .451 .137 .137
    P(ID) #1: (.105 .116 .320 .455 .000 .000)

    PLANE 0 R16: .022 .022 .454 .454 .022 .022
    P(ID) #0: (.009 .009 .265 .715 .000 .000)

    PLANE 0 R17: .128 .052 .341 .341 .062 .062
    P(ID) #0: (.003 .001 .269 .725 .000 .000)

    PLANE 1 R16: .022 .022 .454 .454 .022 .022
    P(ID) #1: (.006 .007 .407 .578 .000 .000)

    PLANE 1 R17: .128 .062 .341 .341 .062 .062
    P(ID) #1: (.002 .001 .411 .584 .000 .000)

(time 90)

    time = 90

(*fact*  plane0  'delta-ROC      6        .95)
(*fact*  plane1  'delta-ROC      7        .95)

(*fact*  plane0  'MRSIG    'afterburner  .80)
(*fact*  plane1  'MRSIG    'afterburner  .80)

(assert-data  plane0  LRSIG     SU2       .85)

    PLANE 0 lr_sig :  su2=.850
    P(ID) #0: (.003 .001 .211 .782 .000 .000)

(assert-data  plane1  LRSIG     SU1       .90)

    PLANE 1 lr_sig :  su1=.900
    P(ID) #1: (.002 .001 .406 .589 .000 .000)

(assert-data  plane0  ENG       dual      .65)

    PLANE 0 #eng   :  eng2=.650
    P(ID) #0: (.000 .000 .197 .801 .000 .000)
```

Rules fire with assertion vectors favoring fighters

Rule 8 really boosts fighter

Prob(ID) exceeds threshold, so cyclical rules start up

PLANE0 ID more certain, but PLANE1 ID undecided yet

New time frame

Detect AB, characteristic of jet craft

Finally engine sensor is correct, #eng = 2

```
(assert-data  plane1  ENG       dual      .70)

    PLANE 1 #eng    :  eng2=.700
    P(ID) #1: (.000 .000 .384 .615 .000 .000)

(assert-data  plane0  IR-exh   2200      .75)

    PLANE 0 ir_ex_t:  <2200=.750
    P(ID) #0: (.000 .000 .181 .818 .000 .000)
(assert-data  plane1  IR-exh   2200      .75)

    PLANE 1 ir_ex_t:  <2200=.750
    P(ID) #1: (.000 .000 .349 .650 .000 .000)

(assert-data  plane0  Radar-sig Foxfire  .45)

    PLANE 0 r-sig   :  foxf=.450
    P(ID) #0: (.000 .000 .419 .580 .000 .000)

(assert-data  plane1  Radar-sig R27       .55)

    PLANE 1 r-sig   :  r27=.550
    P(ID) #1: (.000 .000 .128 .871 .000 .000)

(assert-data  plane0  Width     W3        .50)

    PLANE 0 wid     :  >70=.500
    P(ID) #0: (.000 .000 .419 .580 .000 .000)

(assert-data  plane1  Width     W3        .45)

    PLANE 1 wid     :  >70=.450
    P(ID) #1: (.000 .000 .128 .871 .000 .000)

(assert-data  plane0  Len       L1        .65)

    PLANE 0 len     :  <70=.650
    P(ID) #0: (.000 .000 .431 .568 .000 .000)

(assert-data  plane1  Len       L1        .60)

    PLANE 1 len     :  <70=.600
    P(ID) #1: (.000 .000 .132 .867 .000 .000)

    PLANE 0 R8 : .137 .137 .451 .451 .137 .137
    P(ID) #0: (.000 .000 .431 .568 .000 .000)

    PLANE 1 R8 : .137 .137 .451 .451 .137 .137
    P(ID) #1: (.000 .000 .132 .867 .000 .000)
```

IR temp confirms afterburn indication

Still not sure whether PLANE0 is a MiG-29 or Su-27

As range decreases, getting length and width info

```
    PLANE 0 R16: .022 .022 .454 .454 .022 .022
    P(ID) #0: (.000 .000 .431 .568 .000 .000)

    PLANE 0 R17: .128 .062 .341 .341 .062 .062
    P(ID) #0: (.000 .000 .431 .568 .000 .000)

    PLANE 1 R16: .022 .022 .454 .454 .022 .022
    P(ID) #1: (.000 .000 .132 .867 .000 .000)

    PLANE 1 R17: .128 .062 .341 .341 .062 .062
    P(ID) #1: (.000 .000 .132 .867 .000 .000)


(time 120)

    time = 120

(*fact*  plane0  'delta_IR      50        .65)
(*fact*  plane1  'delta_IR      35        .65)

(assert-data  plane0  MRSIG     AB        .65)

    PLANE 0 mr_sig :  ab=.650
    P(ID) #0: (.000 .000 .425 .574 .000 .000)

(assert-data  plane1  MRSIG     AB        .55)

    PLANE 1 mr_sig :  ab=.550
    P(ID) #1: (.000 .000 .133 .866 .000 .000)

(assert-data  plane0  ENG       dual      .95)

    PLANE 0 #eng   :  eng2=.950
    P(ID) #0: (.000 .000 .420 .579 .000 .000)

(assert-data  plane1  ENG       dual      .95)

    PLANE 1 #eng   :  eng2=.950
    P(ID) #1: (.000 .000 .131 .868 .000 .000)

(assert-data  plane0  Radar-sig Foxfire   .75)

    PLANE 0 r-sig  :  foxf=.750
    P(ID) #0: (.000 .000 .724 .275 .000 .000)

(assert-data  plane1  Radar-sig R27       .85)

    PLANE 1 r-sig  :  r27=.850
    P(ID) #1: (.000 .000 .032 .967 .000 .000)
```

Similar radar returns as before only becoming more confident; This would make a difference if there were more poss. IDs

Finally, greater conf in radar-type IDs PLANE0

```
(assert-data  plane0  Width     W2        .85)

    PLANE 0 wid     :  <70=.850
    P(ID) #0: (.000 .000 .724 .275 .000 .000)

(assert-data  plane1  Width     W2        .90)

    PLANE 1 wid     :  <70=.900
    P(ID) #1: (.000 .000 .032 .967 .000 .000)

(assert-data  plane0  Len     L1          .95)

    PLANE 0 len     :  <70=.950
    P(ID) #0: (.000 .000 .736 .263 .000 .000)

(assert-data  plane1  Len     L1          .95)

    PLANE 1 len     :  <70=.950
    P(ID) #1: (.000 .000 .034 .965 .000 .000)

    PLANE 0 R16: .022 .022 .454 .454 .022 .022
    P(ID) #0: (.000 .000 .736 .263 .000 .000)

    PLANE 0 R17: .128 .062 .341 .341 .062 .062
    P(ID) #0: (.000 .000 .736 .263 .000 .000)

    PLANE 1 R16: .022 .022 .454 .454 .022 .022
    P(ID) #1: (.000 .000 .034 .965 .000 .000)

    PLANE 1 R17: .128 .062 .341 .341 .062 .062
    P(ID) #1: (.000 .000 .034 .965 .000 .000)

(time 150)

    time = 150                                          New time frame

(assert-data  plane0  SIG       SIG3      .45)        SIG info helps to confirm

    PLANE 0 sig     :  sig3=.450
    P(ID) #0: (.000 .000 .964 .035 .000 .000)

(assert-data  plane1  SIG       SIG4      .65)        Remainder of sensor info serves
                                                      to confirm ID; not too helpful
    PLANE 1 sig     :  sig4=.650                      with only six possible IDs, but
    P(ID) #1: (.000 .000 .011 .988 .000 .000)         with more they would be useful

(assert-data  plane0  Radar-sig Foxfire   .95)

    PLANE 0 r-sig   :  foxf=.950
    P(ID) #0: (.000 .000 .993 .006 .000 .000)
```

```
(assert-data  plane1  Radar-sig R27       .95)

    PLANE 1 r-sig   :  r27=.950
    P(ID) #1: (.000 .000 .003 .996 .000 .000)

(assert-data  plane0  Width     W2        .90)

    PLANE 0 wid     :  <70=.900
    P(ID) #0: (.000 .000 .993 .006 .000 .000)

(assert-data  plane1  Width     W2        .95)

    PLANE 1 wid     :  <70=.950
    P(ID) #1: (.000 .000 .003 .996 .000 .000)

(assert-data  plane0  Len       L1        .99)

    PLANE 0 len     :  <70=.990
    P(ID) #0: (.000 .000 .993 .006 .000 .000)

(assert-data  plane1  Len       L1        .99)

    PLANE 1 len     :  <70=.990
    P(ID) #1: (.000 .000 .003 .996 .000 .000)

    PLANE 0 R16: .022 .022 .454 .454 .022 .022
    P(ID) #0: (.000 .000 .993 .006 .000 .000)

    PLANE 0 R17: .128 .062 .341 .341 .062 .062
    P(ID) #0: (.000 .000 .993 .006 .000 .000)

    PLANE 1 R16: .022 .022 .454 .454 .022 .022
    P(ID) #1: (.000 .000 .003 .996 .000 .000)

    PLANE 1 R17: .128 .062 .341 .341 .062 .062
    P(ID) #1: (.000 .000 .003 .996 .000 .000)

(time 180)

    time = 180                                          Last frame

(assert-data  plane0  SIG       SIG3      .95)

    PLANE 0 sig     :  sig3=.950
    P(ID) #0: (.000 .000 .998 .001 .000 .000)

(assert-data  plane1  SIG       SIG4      .95)

    PLANE 1 sig     :  sig4=.950
    P(ID) #1: (.000 .000 .001 .998 .000 .000)
```

```
PLANE 0 R16: .022 .022 .454 .454 .022 .022
P(ID) #0: (.000 .000 .998 .001 .000 .000)


PLANE 0 R17: .128 .062 .341 .341 .062 .062
P(ID) #0: (.000 .000 .998 .001 .000 .000)

PLANE 1 R16: .022 .022 .454 .454 .022 .022
P(ID) #1: (.000 .000 .001 .998 .000 .000)

PLANE 1 R17: .128 .062 .341 .341 .062 .062
P(ID) #1: (.000 .000 .001 .998 .000 .000)


---------- end of trace ----------
```

View of ARF Frames

To give an idea of the other types of frame objects used in OAS1, views of STM, FACT, and airborne-object frames after the sample run are shown below. F.157 is an STM frame object frame, upon which FACTs are asserted.

```
> (view 'f.157)

F.157
    SELF ..... (F.157 () () () (#{Procedure 67}))
    CLASS ..... (STM () () () ())
    PARENTS ..... ()
    DOC ..... ("Basic blackboard - list of objects." () () () ())
    STM ..... (((F.202 . 0.8) (F.216 . 0.65)) () () () ())
    STM-DOC ..... (() () () () ())
    TRACE ..... (OFF () () () ())
```

In the STM slot of F.157 are two FACTs, F.202 and F.216, which were not used during the sample run. Each has an associated confidence which is used along with the belief-factor of RULEs in computing the confidence with which to assert additional FACTs or ID information. If we view the two facts, we see:

```
> (view 'f.202)

F.202
    SELF ..... (F.202 () () () ())
    CLASS ..... (FACT () () () ())
    PARENTS ..... ()
    OBJECT ..... (PLANE0 () () () ())
    ATTRIBUTE ..... (MRSIG () () () ())
    VALUE ..... (AFTERBURNER () () () ())

> (view 'f.216)

F.216
    SELF ..... (F.216 () () () ())
    CLASS ..... (FACT () () () ())
    PARENTS ..... ()
    OBJECT ..... (PLANE0 () () () ())
    ATTRIBUTE ..... (DELTA_IR () () () ())
    VALUE ..... (50 () () () ())
```

FACT frames have **object, attribute** and **value** slots; these two happen to hold information about the first detected airborne object, PLANE0.

Graphics screens

Figures 6 and 7 on the next two pages are copies of computer screens made during the scenario run above. The graphics front-end was added to OAS1 for demonstration purposes, to show the type of information a pilot might like to see in a cockpit display. The graphics makes it especially easy to see how planes are grouped into formations and what their IDs are at a glance. Transitions in probability of ID can be seen as the expert system receives additional sensor data.

For demonstration purposes, each of the two objects in the above scenario represents a formation of eight planes in the screens below. Each of the eight planes is assumed to have the same characteristics. The graph at the right of the screen shows probability of correct ID over time. The "dip" in the curve represents the deliberate "twists" in the trackfile sensor data, intended to make the expert system believe the objects were bombers or transports, instead of fighters.

FIGURE 6: OAS1 SAMPLE SCREEN 1

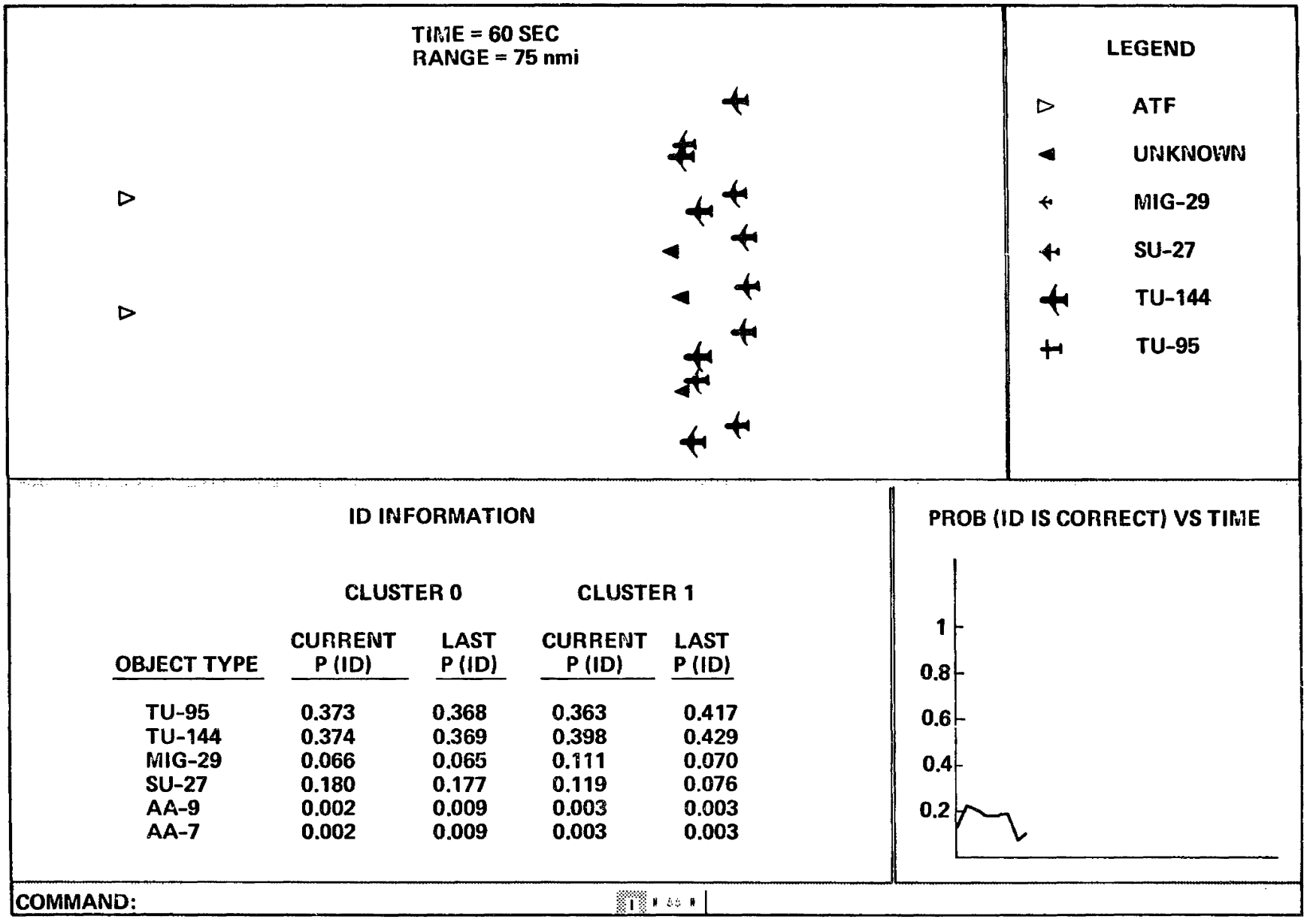
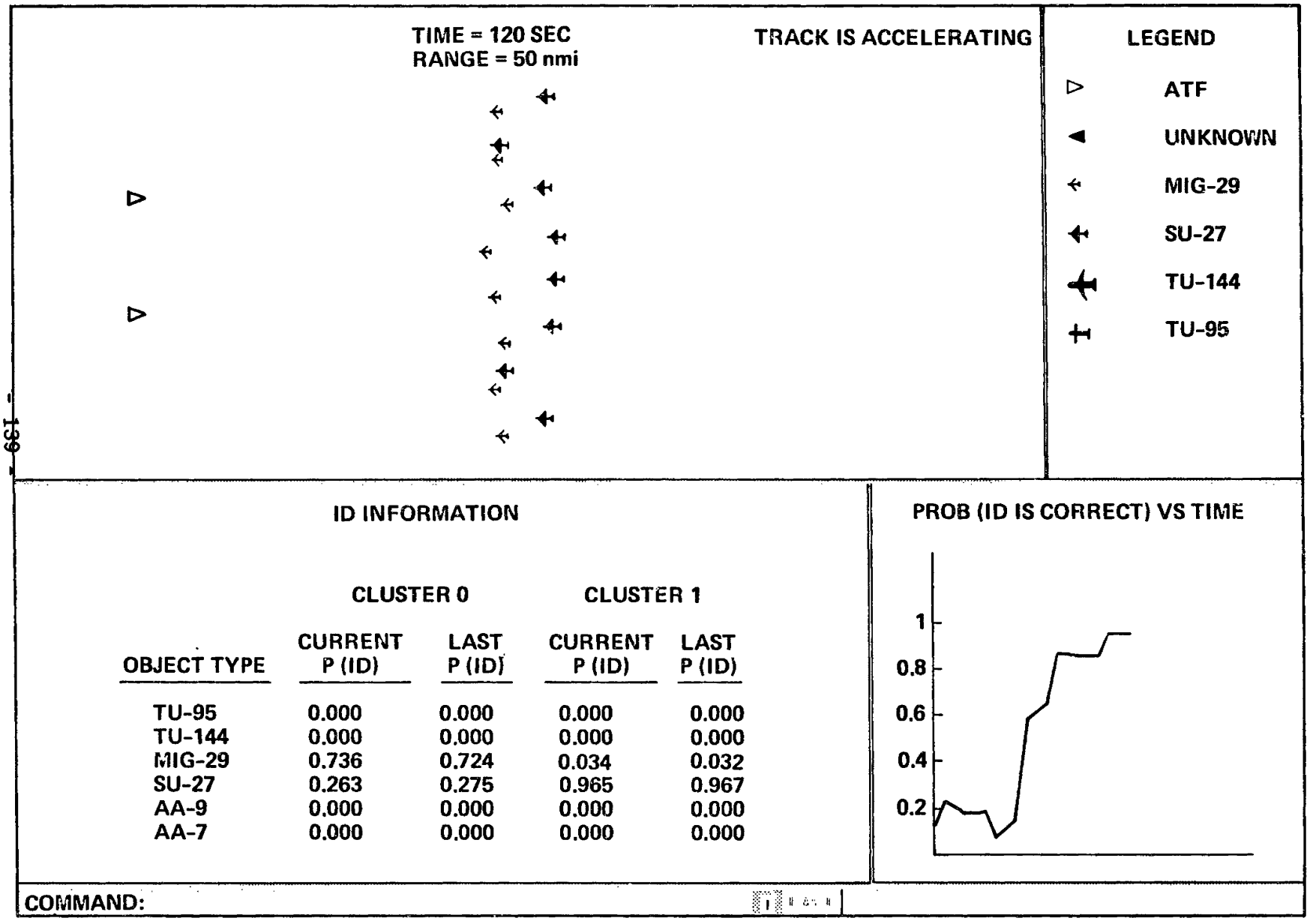


FIGURE 7: OASI SAMPLE SCREEN 2



APPENDIX B

Overview

This appendix contains a sample run of OAS2, the expert system described in Chapter 5. To make it easier to understand the operation of the system, listings have been split into four major sections: **Input Trackfile**, **System Output**, **Demon Operation**, and **Internal Knowledge Representation**.

The "driver" of OAS2 is the input trackfile, which consists of entries that simulate the detection of airborne objects and the reception of sensor returns upon those objects. Listed below are the four different kinds of functions a trackfile can perform, and the effect they have on system operation.

Function: (**detect** obj-name)

Execution of this code indicates that a new airborne object has been detected. As a result of this, an empty **airborne-object frame** is instantiated, and given the label **obj-name**. From this point forward, the frame can be referred to by this label. See the **Internal Knowledge Representation** section for full-length examples of **airborne-object** frames that have had sensor returns asserted upon them.

Function: (**fput** obj-name attribute value)

Execution of this code puts **value** in the **attribute** slot of **obj-name**, triggering any **put** procedural attachments associated with the slot. These procedural attachments often **spawn** demons representing rules that may fire depending upon **value**. See the **Internal Knowledge Representation** section for examples of frames with sensor returns that have been **fput** into slots. See the **Demon Operation** section for a "trace" of the spawning, execution, and killing of demons. The trace is not an explicit output of OAS2, but can be optionally enabled to see the frequency with which demons (rules) are used.

Function: (**vput** obj-name attribute value)

This function is similar to **fput**, except that procedural attachments associated with **slot** are not triggered. A **vput** is used in the trackfile for efficiency purposes only, to ensure that the same demon is not spawned twice in the same time frame. For instance, one demon discussed above uses velocity and radar cross-section values in its **test** slot. If new information is received about both of these attributes in the same time frame, the demon will be spawned twice unless a **vput** is used. This "kludge" would be eliminated in an operational system, where a preprocessor would examine sensor returns to inhibit repeat spawnings of a demon.

Function: (invoke-demon-driver *wm*)

Execution of this expression, which occurs at the end of each time frame, causes the demon inference engine to be invoked. The inference engine takes the next available spawned demon from the queue *wm* and evaluates its **test** slot. If the result is true, the demon's **+act** slot is executed, else the **-act** slot is executed. The results of **+act** slot execution can be: (1) an update of the probability of ID of the object upon which the demon was spawned (belief-maintenance), (2) the spawning of another demon (forward-chaining), (3) an **fput** into the slot of another frame object (knowledge-gathering), or (4) the printing of formation, target, or request information for the user (message-passing).

See the **Demon Operation** section to see the "life-history" of demons as they are spawned, executed, and killed. To see (1), described in the previous paragraph, look at the **Belief-Maintenance** section in **Chapter 5**. To see (2), look at the cyclical rule set demons in the **Demon Operation** section. To see (3), look at the **formation** frames in the **Internal Knowledge Representation** section. To see (4), look at the requests and target information in the **System Output** section.

Function: (spawn-end-of-frame-demons)

This last function spawns all demons associated with end-of-frame tasks. For example, at the end of each frame, we want to group detected airborne objects into formations and determine their possible targets. This function causes the appropriate demons to be spawned.

For an example of the messages produced, see the **System Output** section.

The sections below discuss system operation in a little more detail. Desirable enhancements to the system are mentioned, and Figure 8 at the end of the appendix shows a sample screen of OAS2.

Input Trackfile

Shown below is a listing of a demonstration input trackfile for OAS2. It is similar in format to those used for OAS1, except asserted sensor returns have no confidence values associated with them. In the current version of OAS2, demons that perform belief-maintenance have no provision for confidence propagation.

The scenario represented by the sequence of sensor facts on the next few pages is an incoming enemy formation of two bombers (plane1, plane2) and three fighters (plane3, plane4, plane5). Initially, only the (larger) bombers are detected, but as the fighters close in, they are seen also. Towards the end of the scenario, one of the fighters fires a missile at ownship.

The trackfile that produced results for this appendix was actually seven time "frames" long, in thirty-second intervals from time = 0 to time = 180 seconds. Only the first four or five frames are shown here, however, because for such a small scenario, the later frames have minimal effect on the determination of identity. Furthermore, repetitive fact assertions have been omitted for brevity. **Velocity** and **altitude**, for example, are

asserted for each airborne object in *every* trackfile frame when the system is actually run, but are only shown here the first time. Explanatory comments are shown in *italics*.

OAS2 Input Trackfile Listing

```
(TIME 0)

(detect plane1)                          ; detect two airborne objects; call
(detect plane2)                          ; them plane1 and plane2

(vput plane1 'xy-pos '(100 3))           ; x and y positions in nmi, relative
(vput plane2 'xy-pos '(100 1))           ; to ownship position (0,0)

(fput plane1 'velocity 0.9)              ; velocity = Mach 0.9
(fput plane2 'velocity 0.95)

(vput plane1 'altitude 20000)            ; altitude = 20,000 feet
(vput plane2 'altitude 21000)

(vput plane1 'RCS 'medium)               ; RCS = radar cross-section
(vput plane2 'RCS 'medium)

(spawn-end-of-frame-demons)              ; spawn formation and target demons

(invoke-demon-driver *wm*)               ; run demons on agenda *wm*


(TIME 30)                                ; new time frame

(fput plane1 'xy-pos '(88 0.1))          ; planes are getting closer
(fput plane2 'xy-pos '(88 1))

(fput plane1 'altitude 22500)            ; altitude increasing
(fput plane2 'altitude 23000)

(fput plane1 'velocity 0.85)             ; velocity decreasing
(fput plane2 'velocity 0.90)

(vput plane1 'RCS 'large)                ; large RCS is
(vput plane2 'RCS 'large)                ; characteristic of a bomber

(fput plane1 'LRSIG 'Soviet-class1)
(fput plane2 'LRSIG 'Soviet-class1)

(spawn-end-of-frame-demons)
(invoke-demon-driver *wm*)


(TIME 60)

(detect plane3)                          ; close enough now to see three
(detect plane4)                          ; other planes
(detect plane5)
```

```
(fput plane1 'xy-pos '(75 -1))
(fput plane2 'xy-pos '(75 -0.5))
(vput plane3 'xy-pos '(76 1.5))          give their positions
(vput plane4 'xy-pos '(75 -0.4))
(vput plane5 'xy-pos '(74 1))

(fput plane3 'velocity 1.52)             they are flying much faster than
(fput plane4 'velocity 1.55)             plane1 and plane2
(fput plane5 'velocity 1.47)
(vput plane3 'altitude 30000)            they're at higher altitude, too
(vput plane4 'altitude 30000)
(vput plane5 'altitude 30000)

(fput plane3 'LRSIG 'Soviet-class2)      they have a different "SIG" than
(fput plane4 'LRSIG 'Soviet-class2)      plane1 and plane2
(fput plane5 'LRSIG 'Soviet-class2)

(fput plane1 'eng 4)                     characteristic of a bomber
(fput plane2 'eng 4)
(fput plane3 'eng 2)                     characteristic of a fighter
(fput plane4 'eng 2)
(fput plane5 'eng 2)

(fput plane3 'IR_temp 950)               infrared temperature in degrees
(fput plane4 'IR_temp 975)
(fput plane5 'IR_temp 950)

(spawn-end-of-frame-demons)
(invoke-demon-driver *wm*)


(TIME 90)

(detect rocket1)                         another airborne object is detected;
                                         give it the name rocket1

(vput rocket1 'xy-pos '(57 -0.7))
(fput rocket1 'IR_temp 2500)             much hotter than other objects
(fput rocket1 'velocity 2.1)             much faster than other objects

(vput rocket1 'altitude 32500)           same altitude as the fighters

(vput plane3 'RCS 'medium)
(vput plane4 'RCS 'medium)
(vput plane5 'RCS 'medium)

(fput plane3 'IR_temp 1500)              temperature of engines increasing
(fput plane4 'IR_temp 1450)
(fput plane5 'IR_temp 1525)
```

```
(fput plane1 'radar 'none)              no radar info on bombers
(fput plane2 'radar 'none)
(fput plane3 'radar 'foxfire)           two different radars for fighters
(fput plane4 'radar 'hi-lark)
(fput plane5 'radar 'hi-lark)

(fput plane3 'maneuvers 's-turns)       two of the fighters are executing
(fput plane5 'maneuvers 's-turns)       weaving maneuvers

(spawn-end-of-frame-demons)
(invoke-demon-driver *wm*)


(TIME 120)

(fput rocket1 'xy-pos '(28 -0.4))       rocket closing quickly
                                        on ownship

(fput rocket1 'altitude 25000)          reaching ownship's altitude

(fput plane4 'maneuvers 's-turns)

(vput rocket1 'RCS 'small)

(fput plane1 'length 85)                at close range, begin to receive
(fput plane1 'width  75)                length and width information
(fput plane2 'length 80)
(fput plane2 'width  75)

(fput plane3 'width 45)
(fput plane4 'width 45)
(fput plane5 'width 45)

(spawn-end-of-frame-demons)
(invoke-demon-driver *wm*)

(TIME 150)                              other similar time frames follow

      .
      .
      .
```

System Output

This section contains a listing of OAS2 output generated when the input trackfile is loaded. The user is notified of airborne object detections, constituency of formations, and possible targets of incoming planes. In anticipation of the time when OAS2 interfaces with a sensor manager, the system also provides requests for sensor data if availability of that data would enable a rule to "test" its if-part. Finally, OAS2 notifies the user when invocation of the cyclical rule set begins. For an airborne object with n possible IDs, this occurs when Prob(identity is ID_i) > (3/(2 * n)), for $1 \leq i \leq n$. For an airborne object that's either a fighter or a bomber, for example, Prob(ID is fighter) or Prob(ID is bomber) would have to exceed 0.75 for invocation of the cyclical rule set to occur. This threshold is somewhat arbitrary, and can easily be changed when more research is done.

A major weakness of OAS2 is its current inability to notify the user of the most likely identity of each airborne object, without stopping the run and examining the airborne object frames. Once a more sophisticated belief mechanism is installed, an unobtrusive way to notify the user of the current ID must be developed.

```
> (load '~demons/track)
;Loading "~demons/track.t" into *SCRATCH-ENV*

Time is now 0.

Detected an airborne object...                    result of (detect plane1)
Detected an airborne object...

Time is now 30.

Prob(PLANE1 = BOMBER) exceeds threshold of 3/4    cyclical rules are invoked
 -- invoking cyclical rules.                      (both plane1 and plane2 have
Prob(PLANE2 = BOMBER) exceeds threshold of 3/4    two possible IDs at this point)
 -- invoking cyclical rules.

Now grouping planes into formations...            Formation demon invoked...

[Binding FORMATION#.175]                          one formation found

FORMATION#.175 consists of (PLANE2 PLANE1).       Both planes are near enough
Possible targets include:                         to be in a single formation

     NAME:     ATF                                These are targets created with
     LOCATION: (0.0 0.0)                          the create-base macro described
     VALUE:    6                                  in Chapter 5

     NAME:     H.Q.
     LOCATION: (-700.0 -200.0)
     VALUE:    10


Time is now 60.

Detected an airborne object...                    three fighters detected
Detected an airborne object...
Detected an airborne object...

Need RCS value of PLANE3                          a request for sensor info;
Need RCS value of PLANE4                          a rule would have fired if RCS
                                                  had been available

Prob(PLANE1 = BOMBER) exceeds threshold of 1/2    only three of the five planes
 -- invoking cyclical rules.                      have Prob(ID) high enough so
Prob(PLANE2 = BOMBER) exceeds threshold of 1/2    that the cyclical rules are
 -- invoking cyclical rules.                      invoked
Prob(PLANE5 = FIGHTER) exceeds threshold of 3/4
 -- invoking cyclical rules.
```

```
Now grouping planes into formations...

[Binding FORMATION#.304]                          Two formations this time
[Binding FORMATION#.307]
FORMATION#.304 consists of (PLANE2 PLANE1).
Possible targets include:

     NAME:     ATF
     LOCATION: (0.0 0.0)
     VALUE:    6

     NAME:     H.Q.
     LOCATION: (-700.0 -200.0)
     VALUE:    10

     NAME:     SAM3                           formation's heading changed,
     LOCATION: (-690.0 50.0)                  so another target became
     VALUE:    8                              possible

FORMATION#.307 consists of (PLANE3 PLANE4 PLANE5).
Possible targets include:

     NAME:     ATF                            second formation's targets
     LOCATION: (0.0 0.0)
     VALUE:    6

     NAME:     H.Q.
     LOCATION: (-700.0 -200.0)
     VALUE:    10

     NAME:     SAM3
     LOCATION: (-690.0 50.0)
     VALUE:    8


Time is now 90.

Detected an airborne object...                    sixth object detected;
                                                  this is the rocket

PLANE3 is exhibiting S-TURNS maneuvers.           weaving maneuvers of two
PLANE5 is exhibiting S-TURNS maneuvers.           fighters detected

Need RCS value of ROCKET1
Need RCS value of ROCKET1

Possible mission of PLANE3 is ESCORT              possible mission deduced
Possible mission of PLANE5 is ESCORT              from maneuver information

Prob(PLANE1 = BOMBER) exceeds threshold of 1/2
 -- invoking cyclical rules.
```

```
Prob(PLANE2 = BOMBER) exceeds threshold of 1/2
 -- invoking cyclical rules.
Prob(PLANE5 = FIGHTER) exceeds threshold of 3/4
 -- invoking cyclical rules.

The phase of PLANE3's mission is DASH          mission phase deduced from
The phase of PLANE4's mission is DASH          acceleration information
The phase of PLANE5's mission is DASH

Now grouping planes into formations...

[Binding FORMATION#.451]                       still two formations
[Binding FORMATION#.454]

FORMATION#.451 consists of (PLANE2 PLANE1).
Possible targets include:

     NAME:     ATF
     LOCATION: (0.0 0.0)
     VALUE:    6

     NAME:     H.Q.
     LOCATION: (-700.0 -200.0)
     VALUE:    10

     NAME:     SAM3
     LOCATION: (-690.0 50.0)
     VALUE:    8

FORMATION#.454 consists of (ROCKET1 PLANE3 PLANE4 PLANE5).
Possible targets include:

     NAME:     ATF                              rocket1 considered part of
     LOCATION: (0.0 0.0)                        this formation because it is
     VALUE:    6                                still close to fighter that
                                                fired it
     NAME:     H.Q.
     LOCATION: (-700.0 -200.0)
     VALUE:    10

     NAME:     SAM3
     LOCATION: (-690.0 50.0)
     VALUE:    8

Time is now 120.

PLANE4 is exhibiting S-TURNS maneuvers.

The phase of ROCKET1's mission is DESCENT      rocket descending to ownship's
                                               altitude
```

```
Possible mission of PLANE4 is ESCORT

Prob(PLANE5 = FIGHTER) exceeds threshold of 3/4
 -- invoking cyclical rules.

Now grouping planes into formations...

[Binding FORMATION#.515]                          three formations now: rocket
[Binding FORMATION#.518]                          has accelerated away
[Binding FORMATION#.521]

FORMATION#.515 consists of (PLANE2 PLANE1).
Possible targets include:

     NAME:      ATF
     LOCATION:  (0.0 0.0)
     VALUE:     6

     NAME:      SAM3
     LOCATION:  (-690.0 50.0)
     VALUE:     8

FORMATION#.518 consists of (PLANE5 PLANE3 PLANE4).
Possible targets include:

     NAME:      ATF
     LOCATION:  (0.0 0.0)
     VALUE:     6

     NAME:      H.Q.
     LOCATION:  (-700.0 -200.0)
     VALUE:     10

FORMATION#.521 consists of (ROCKET1).
Possible targets include:

     NAME:      ATF                               rocket heading towards ownship
     LOCATION:  (0.0 0.0)
     VALUE:     6

     NAME:      SAM3
     LOCATION:  (-690.0 50.0)
     VALUE:     8

                .                                 for small scenario, additional
                .                                 frames provide little insight
                .

Done.
```

Demon Operation

This listing shows the optional system output available when the tracing facility for demons is enabled. Demon spawns, executes, and kills are shown in the order they occur. In addition, meta-demons are identified when they are used.

For tracing purposes, demons are divided into four major groups, depending on the type of rules they represent. **Physical Attribute** demons deal with physical properties of airborne objects, such as velocity, altitude, number of engines, length and width. **Mission-Leval Attribute** demons handle formation, target, mission, and ID-oriented tasks. **Emissions** demons deal with radiated properties of airborne objects, such as electromagnetic (radar), infrared, and "signature" emissions. Finally, **Action** demons handle airborne object maneuvers.

A few representative demons from each of the four groups are shown below.

Physical Attribute Demon Trace

```
Time is now 0.

    Spawning demon: SPEED-DEMON1.116
    (SPEED-DEMON1 #{Object 73} F.110)

    Spawning demon: SPEED-DEMON2.117
    (SPEED-DEMON2 #{Object 73} F.110)

    Spawning demon: SPEED-DEMON3.118
    (SPEED-DEMON3 #{Object 73} F.110)

    Spawning demon: SPEED-DEMON4.119
    (SPEED-DEMON4 #{Object 73} F.110)

    Killing demon: SPEED-DEMON1.116
    Executing -act of demon: SPEED-DEMON1.116

    Executing Demon: SPEED-DEMON2.117 as
            part of XOR-DEMON.114

Time is now 30.

    Spawning demon: ALT-DEMON.135
    (ALT-DEMON #{Object 73} F.110)

    Killing demon: ALT-DEMON.135
    Executing -act of demon: ALT-DEMON.135

Time is now 60.

    Spawning demon: ENG-DEMON1.219
    (ENG-DEMON1 #{Object 73} F.110)

    Spawning demon: ENG-DEMON2.220
    (ENG-DEMON2 #{Object 73} F.110)

    Killing demon: ENG-DEMON1.219
    Executing -act of demon: ENG-DEMON1.219

    Executing Demon: ENG-DEMON2.220 as
            part of XOR-DEMON.217
```

new velocity and RCS info cause 4 demons, wrapped by an XOR-demon, to spawn; speed-demons relate velocity and RCS to ID

*#{Object 73} is *wm*, the demon queue, and F.110 is the airborne-object frame upon which the demons were spawned*

demon is killed because test slot was false

speed-demon2, however, tests true and fires; remaining demons enclosed by the XOR-demon meta-demon are automatically killed, and no message is printed

alt-demon check rate-of-climb (ROC) and infer ID

ROC not great enough: tests false and dies

eng-demons check number of engines and infer ID

tested true and fired

```
Time is now 90.

    Spawning demon: LEN-DEMON1.469
    (LEN-DEMON1 #{Object 73} F.110)

    Spawning demon: LEN-DEMON2.470
    (LEN-DEMON2 #{Object 73} F.110)

    Spawning demon: WID-DEMON1.473
    (WID-DEMON1 #{Object 73} F.110)

    Spawning demon: WID-DEMON2.474
    (WID-DEMON2 #{Object 73} F.110)

    Spawning demon: WID-DEMON3.475
    (WID-DEMON3 #{Object 73} F.110)

    Killing demon: LEN-DEMON1.469
    Executing -act of demon: LEN-DEMON1.469

    Executing Demon: LEN-DEMON2.470 as
             part of XOR-DEMON.467

    Killing demon: WID-DEMON1.473
    Executing -act of demon: WID-DEMON1.473

    Killing demon: WID-DEMON2.474
    Executing -act of demon: WID-DEMON2.474

    Executing Demon: WID-DEMON3.475 as
             part of XOR-DEMON.471
```

length and width demons test for threshold and assert ID information

```
Time is now 0.

    Spawning demon: ID-DEMON.126
    (ID-DEMON #{Object 73} F.110)

    Spawning demon: FRAME-SPAWN-DEMON.128
    (FRAME-SPAWN-DEMON #{Object 73})

    Killing demon: ID-DEMON.126
    Executing -act of demon: ID-DEMON.126

    Executing demon: FRAME-SPAWN-DEMON.128

    Spawning demon: FORMATION-DEMON.131
    (FORMATION-DEMON #{Object 73})

    Killing demon: FRAME-SPAWN-DEMON.128

    Executing demon: FORMATION-DEMON.131

    Spawning demon: TARGET-DEMON.134
    (TARGET-DEMON #{Object 73} F.133)

    Killing demon: FORMATION-DEMON.131

    Killing demon: TARGET-DEMON.134
    Executing -act of demon: TARGET-DEMON.134

Time is now 30.

    Spawning demon: ID-DEMON.157
    (ID-DEMON #{Object 73} F.110)

    Spawning demon: FRAME-SPAWN-DEMON.159
    (FRAME-SPAWN-DEMON #{Object 73})

    Executing demon: ID-DEMON.157

    Spawning demon: F-CR-DEMON.164
    (F-CR-DEMON #{Object 73} F.110 BOMBER)

    Spawning demon: B-CR-DEMON.165
    (B-CR-DEMON #{Object 73} F.110 BOMBER)

    Spawning demon: T-CR-DEMON.166
    (T-CR-DEMON #{Object 73} F.110 BOMBER)

    Executing demon: FRAME-SPAWN-DEMON.159
```

These demons are spawned at end-of-frame; ID-demon is the "root" demon for the cyclical rule set: it checks to see if Prob(ID) is past threshold, and spawns other demons if it is. This time frame it is not.

Frame-spawn demon spawns the demons associated with the end of frame; here we see it spawns Formation-demon, then gets killed.

Formation demon trys to group airborne objects and spawns target-demon, which looks for their targets.

ID-demon spawned again--this time-frame the cyclical rules will be invoked.

spawned again

Prob(ID) > thresh, so three demons spawned with "bomber" (the current ID) as argument; Demons check ID and spawn comrad-demon to check CR.

F-CR: sees if ID is fighter
B-CR: sees if ID is bomber
T-CR: sees if ID is transport

Frame-spawn-demon's turn to test and execute.

```
Spawning demon: FORMATION-DEMON.172
(FORMATION-DEMON #{Object 73})

Killing demon: ID-DEMON.157

Killing demon: FRAME-SPAWN-DEMON.159

Killing demon: F-CR-DEMON.164
Executing -act of demon: F-CR-DEMON.164

Executing Demon: B-CR-DEMON.165 as
        part of XOR-DEMON.162

Spawning demon: COMRAD-DEMON.173
(COMRAD-DEMON #{Object 73} F.110)

Executing demon: FORMATION-DEMON.172

Spawning demon: TARGET-DEMON.177
(TARGET-DEMON #{Object 73} F.176)

Killing demon: FORMATION-DEMON.172

Executing demon: COMRAD-DEMON.173

Spawning demon: POO-DEMON.178
(POO-DEMON #{Object 73} F.110)

Executing demon: TARGET-DEMON.177

Spawning demon: SAYFORMS-DEMON.180
(SAYFORMS-DEMON #{Object 73} F.176)

Killing demon: COMRAD-DEMON.173

Killing demon: TARGET-DEMON.177

Executing demon: POO-DEMON.178

Executing demon: SAYFORMS-DEMON.180

Killing demon: POO-DEMON.178

Killing demon: SAYFORMS-DEMON.180
```

ID-demon & frame-spawn-demon previously spawned are killed

Now back to CR demons; F-CR killed because ID is bomber;

B-CR executes though; spawns comrad-demon, which looks for combat radius of a bomber.

Formation demon executes, and spawns target demon.

Execution of comrad-demon causes POO-demon (point-of-origin) to spawn; checks for base object might be from.

Go back and execute target-demon, which spawns sayforms-demon, whose purpuse is to print out formations and targets.

execute POO-demon, asserting new Prob(ID) info based on enemy deployment of planes at bases.

print out formations, targets

```
Time is now 0.

Time is now 30.

    Spawning demon: LRSIG-DEMON1.151
    (LRSIG-DEMON1 #{Object 73} F.110)

    Spawning demon: LRSIG-DEMON2.152
    (LRSIG-DEMON2 #{Object 73} F.110)

    Executing Demon: LRSIG-DEMON1.151 as
             part of XOR-DEMON.149

    Executing Demon: LRSIG-DEMON1.152 as
             part of XOR-DEMON.150

Time is now 60.

    Spawning demon: IR-DEMON1.239
    (IR-DEMON1 #{Object 73} F.181)

    Spawning demon: IR-DEMON2.240
    (IR-DEMON2 #{Object 73} F.181)

    Spawning demon: IR-DEMON3.241
    (IR-DEMON3 #{Object 73} F.181)

    Spawning demon: IR-DEMON4.242
    (IR-DEMON4 #{Object 73} F.181)

    Killing demon: IR-DEMON1.239
    Executing -act of demon: IR-DEMON1.239

    Executing Demon: IR-DEMON2.240 as
             part of XOR-DEMON.237

    Spawning demon: TYPE-DEMON1.269
    (TYPE-DEMON1 #{Object 73} F.181)

    Spawning demon: TYPE-DEMON2.270
    (TYPE-DEMON2 #{Object 73} F.181)

    Spawning demon: TYPE-DEMON3.271
    (TYPE-DEMON3 #{Object 73} F.181)

    Spawning demon: TYPE-DEMON4.272
    (TYPE-DEMON4 #{Object 73} F.181)
```

Long-Range "Signature" demons spawned: infers ID.

IR demons check temp of objects, assert data about engine type or mode, and spawn other demons (type-demons) to infer ID from this

demon executes, fputting info to frame, spawning type-demon

```
    Killing demon: TYPE-DEMON1.269
    Executing -act of demon: TYPE-DEMON1.269

    Executing Demon: TYPE-DEMON2.270 as
            part of XOR-DEMON.267
```

type-demon fires, updating Prob(ID)

```
Time is now 90.

    Spawning demon: RADAR-DEMON1.369
    (RADAR-DEMON1 #{Object 73} F.110)

    Spawning demon: RADAR-DEMON2.370
    (RADAR-DEMON2 #{Object 73} F.110)

    Spawning demon: RADAR-DEMON3.371
    (RADAR-DEMON3 #{Object 73} F.110)

    Killing demon: RADAR-DEMON1.369
    Executing -act of demon: RADAR-DEMON1.369

    Killing demon: RADAR-DEMON2.370
    Executing -act of demon: RADAR-DEMON2.370

    Killing demon: RADAR-DEMON3.371
    Executing -act of demon: RADAR-DEMON3.371
```

radar-demons check radar type of object and update ID

Action Demon Trace

```
Time is now 0.

Time is now 30.

Time is now 60.

Time is now 90.

    Spawning demon: MNUVR-DEMON1.394
    (MNUVR-DEMON1 #{Object 73} F.181)

    Spawning demon: MNUVR-DEMON2.395
    (MNUVR-DEMON2 #{Object 73} F.181)

    Executing Demon: MNUVR-DEMON1.394 as
             part of XOR-DEMON.392

Time is now 120.

    Spawning demon: MNUVR-DEMON1.465
    (MNUVR-DEMON1 #{Object 73} F.183)

    Spawning demon: MNUVR-DEMON2.466
    (MNUVR-DEMON2 #{Object 73} F.183)

    Executing Demon: MNUVR-DEMON1.465 as
             part of XOR-DEMON.463

Time is now 150.

Time is now 180.
```

spawn of these demons doesn't occur until close range, when maneuvers and such can be detected

mnuvr-demon checks maneuvers updates mission or phase of object, and prints message on system output

Internal Knowledge Representation

To conclude this appendix, we show the effect that the trackfile assertions and the demon actions have on the internal representation of airborne objects. The listings show views of **airborne-object**, **formation**, and **hypotheses** frames after a complete system run, affording a demonstration of the representation capabilities of the system.

Typing the variable *plane-list* to the LISP prompt yields a list of the airborne-objects detected to date.

```
> plane-list
(PLANE1 PLANE2 PLANE3 PLANE4 PLANE5 ROCKET1)
```

The variable *form-list* contains a list of the current formations detected.

```
> form-list
(FORMATION#.554 FORMATION#.557 FORMATION#.560)
```

We can now view one of the formations:

```
> (view formation#.554)

F.555
    SELF ..... (F.555 () () () ())
    CLASS ..... (CLUSTER () () () ())
    PARENTS ..... ()
    NAME ..... (FORMATION#.554 () () () ())
    CONSTITUENTS ..... ((PLANE2 PLANE1) () () () (#{Procedure 74}))
    HEADING ..... (180.0 () () () (#{Procedure 75}))
    TARGETS ..... ((ATF SAM3) () () () (#{Procedure 76}))
```

FORMATION#.554 consists of PLANE2 and PLANE1, and its heading is 180 degrees. Possible targets are ATF (ownship) and a surface-to-air missile site, SAM3. Now we can view one of the formation's constituents:

```
> (view plane2)

F.112
    SELF ..... (F.112 () () () (#{Procedure 77}))
    CLASS ..... (AIRBORNE_OBJECT () () () ())
    PARENTS ..... ((F.555) () () () ())
    NAME ..... (PLANE2 () () () ())
    VELOCITY ..... (0.9 () () () (#{Procedure 78}))
    XY-POS ..... ((32 -2.0) () () (#{Procedure 79}) (#{Procedure 80})
    HEADING ..... (180.0 () () () ())
    RCS ..... (LARGE () () () (#{Procedure 81}))
    IR_TEMP ..... (() () () () (#{Procedure 82}))
    RADAR ..... (NONE () () () (#{Procedure 83}))
    LRSIG ..... (SOVIET-CLASS1 () () () (#{Procedure 84}))
    ENG ..... (4 () () () (#{Procedure 85}))
    ENG_TYPE ..... (() () () () (#{Procedure 86}))
    LENGTH ..... (80 () () () (#{Procedure 87}))
    WIDTH ..... (75 () () () (#{Procedure 88}))
    MANEUVERS ..... (() () () () (#{Procedure 89}))
    ALTITUDE ..... (23000 () () (#{Procedure 90}) (#{Procedure 91}))
    PREV-ALTITUDE .... (21000 () () () ())
    HYP-PTR ..... (F.113 () () () ())
    COM_RAD ..... (8000 () () () (#{Procedure 92}))
    BASE_POO ..... ((#{Object 98} #{Object 94} #{Object 95}) () () ()
(#{Procedure93}))
    POSS_IDS ..... ((ROCKET BOMBER TRANSPORT) () () () ())
    PTR_LIST ..... ((F.130 F.161 F.266) () () () ())
    PID_LIST ..... ((0.034 0.479 0.487) () () () ())
    PID%ROCKET ..... ((0.05 0.05 0.0 0.0) () () () ())
    PID%BOMBER ..... ((0.5 0.9 0.8 0.5 0.0 0.0) () () () ())
    PID%TRANSPORT .... ((0.5 0.9 0.8 0.0 0.0) () () () ())
```

Many of the values in plane2's slots were asserted directly from the input trackfile, but quite a few others were added by demon-firings. Note that procedural attachments are represented by #{Procedure x}; most reside in the fifth location of a slot, reserved for the put/after procedural attachment.[67] Note also the contents of the BASE_POO slot: three possible bases where the airborne object might have originated. Information on

---

67. A slot holds five lists, the first of which is for its value, and the remaining four for get/before, get/after, put/before, and put/after procedural attachments, respectively.

Prob(ID) for this object is contained in the final six slots. ID is either a bomber or a transport. Note that Prob(ID is transport) is more likely even though the last three ID assertion confidence levels are the same for both bomber and transport (0.5 0.9 0.8). This is because the order in which assertions were made affects past values of Prob(ID), which are, in turn, used to calculate the current Prob(ID).

If we view one of the fighters, plane5, we see:

```
> (view plane5)

F.185
    SELF ..... (F.185 () () () (#{Procedure 77}))
    CLASS ..... (AIRBORNE_OBJECT () () () ())
    PARENTS ..... ((F.558) () () () ())
    NAME ..... (PLANE5 () () () ())
    VELOCITY ..... (1.7 () () () (#{Procedure 78}))
    XY-POS ..... ((30 -2.5) () () (#{Procedure 79}) (#{Procedure 80})
    HEADING ..... (180.0 () () () ())
    RCS ..... (MEDIUM () () () (#{Procedure 81}))
    IR_TEMP ..... (1525 () () () (#{Procedure 82}))
    RADAR ..... (HI-LARK () () () (#{Procedure 83}))
    LRSIG ..... (SOVIET-CLASS2 () () () (#{Procedure 84}))
    ENG ..... (2 () () () (#{Procedure 85}))
    ENG_TYPE ..... (AFTERBURNER () () () (#{Procedure 86}))
    LENGTH ..... (65 () () () (#{Procedure 87}))
    WIDTH ..... (45 () () () (#{Procedure 88}))
    MANEUVERS ..... (S-TURNS () () () (#{Procedure 89}))
    ALTITUDE ..... (33500 () () (#{Procedure 90}) (#{Procedure 91}))
    PREV-ALTITUDE .... (30000 () () () ())
    HYP-PTR ..... (F.186 () () () ())
    COM_RAD ..... (1000 () () () (#{Procedure 92}))
    BASE_POO ..... ((#{Object 94} #{Object 95}) () () (#{Procedure93})
    POSS_IDS ..... ((ROCKET FIGHTER) () () () ())
    PTR_LIST ..... ((F.261 F.264) () () () ())
    PID_LIST ..... ((0.006 0.994) () () () ())
    PID%ROCKET ..... ((0.05 0.0 0.0) () () () ())
    PID%FIGHTER ..... ((0.9 0.8 0.95 0.7  0.9  0.85 0.8 0.5 0.0 0.0) () ()
() ())
```

Prob(ID) for this object is much more definitive. If we want to know data about this fighter's mission, we look at the HYP-PTR slot above, which

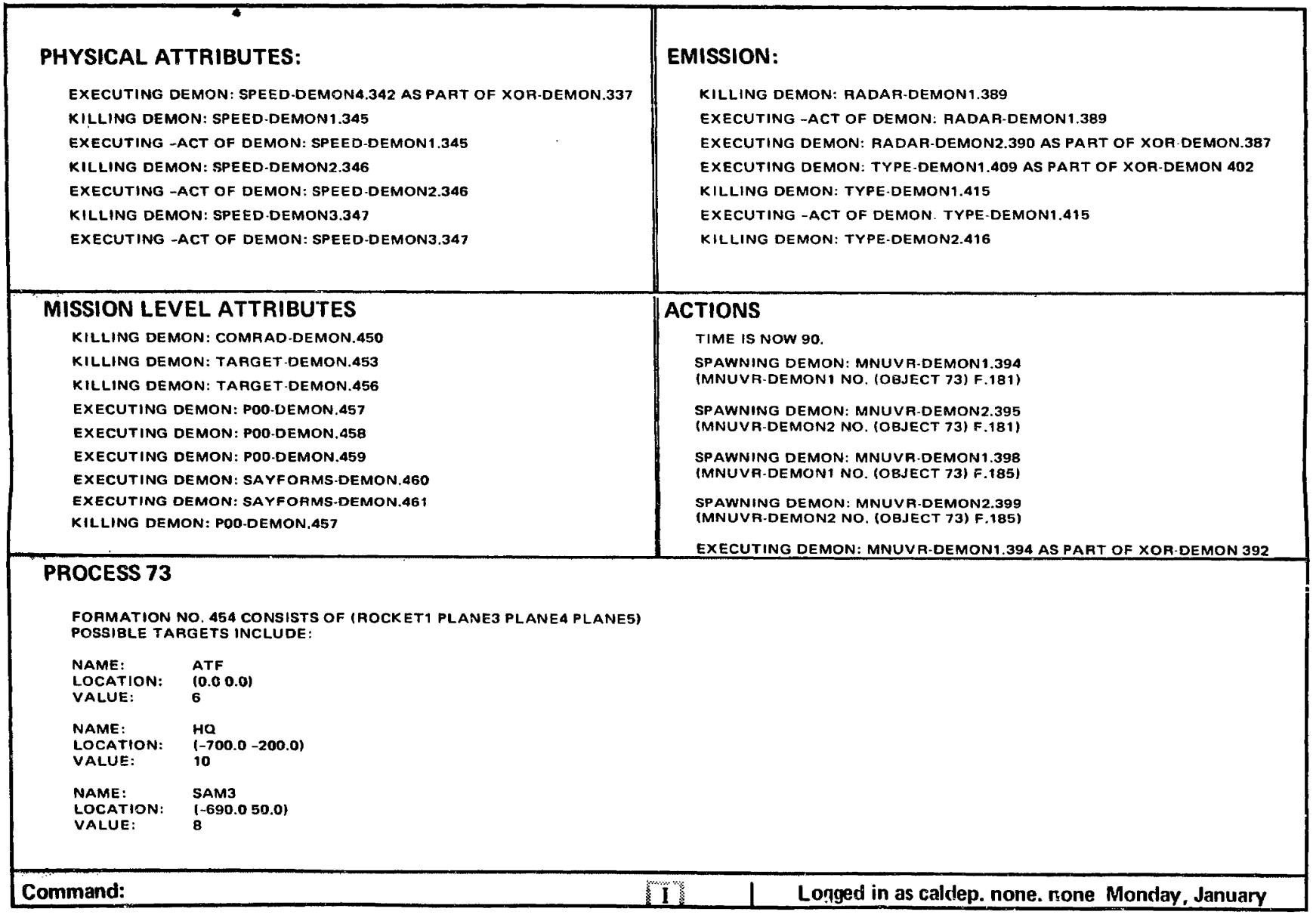
points to F.186. If we view this frame object, we see:

```
> (view 'f.186)

F.186
    SELF ..... (F.186 () () () ())
    CLASS ..... (HYPOTHESES () () () ())
    PARENTS ..... ((F.185) () () () ())
    MISSION ..... (ESCORT () () () (#{Procedure 96}))
    PHASE ..... (DASH () () () (#{Procedure 97}))
    ACC-HIST ..... (() () () () ())
```

The mission of plane5 is **escort**, and the phase of the mission is **dash**. Currently, the **ACC-HIST** (acceleration history) slot of hypotheses is not used.

FIGURE 8. OAS2 SAMPLE SCREEN



REFERENCES

Blackman S. S. and Broida T. J. A Method of Sensor Allocation for a Multitarget-Multisensor Tracking Problem (U), Hughes Aircraft Company, Ref. 2312.10/31, March 1983.

Ben-Basset, M. Multimembership and Multiperspective Classification: Introduction, Applicationsa, and a Bayesian Model. *IEEE Transactions on Systems, Man, and Cybernetics.* Vol. SMC-10, No. 6, pp. 331-336, 1980.

Bobrow D. B. and Winograd T. An Overview of KRL, A Knowledge Representation Language. *Cognitive Science,* Volume 1, No. 1, 1977.

Clancey W. J. Classification Problem Solving. Heuristic Programming Project, Computer Science Department, Stanford University, Stanford, California, 1984.

Charniak E., Riesbeck C. K., and McDermott D. V. *Artificial Intelligence Programming,* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1980.

Dempster A. P. Upper and lower probabilities induced by a multivalued mapping, *Annals of Mathematical Statistics,* Vol. 38, pp.325-339, 1967.

Dillard R. A. Computing Confidences in Tactical Rule-Based Systems by Using Dempster-Shafer Theory. Technical Note 649, Naval Oceans Systems Center, San Diego, California, September 1983.

Dillard R. A. Computing Probability Masses in Rule-Based Systems. Technical Note 545, Naval Oceans Systems Center, San Diego, California, September 1982.

Dolan C. P. Representations for Conceptual Objects: T-CD^2, Discrimination Net, and Demon Control Structures. Tools Note 1, Artificial Intelligence Laboratory, UC Los Angeles, California, June 1984.

Dyer M. G. *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension,* The MIT Press, Cambridge, Massachusetts, 1983.

Duda R. O., Hart P. E., Nilsson N. J. Subjective Bayesian Methods for Rule-Based Inference Systems. Technical Note 124, SRI International, Menlo Park, California, January 1976.

Edwards G. R. *A Rule and Frame System*, version 2.2. Hughes Artificial Intelligence Center, Calabasas, Califronia, April 1984.

Fikes F. and Hendrix G. A Network-Based Knowledge Representation and its Natural Deduction System. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Fox M. S., Lowenfeld S., and Kleinosky P. Techniques for Sensor-Based Diagnosis. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, Karlsruhe, West Germany.

Garvey T. D., Lowrance J. D., Fischler M. A. An Inference Technique for Integrating Knowledge from Disparate Sources. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981, University of British Columbia, Vancouver, Canada.

Geschke M. J., Bullock R. A., and Widmaier L. E. *TAC * II An Expert Knowledge Based System For Tactical Decision Making.* Master's Thesis, Naval Postgraduate School, Monterey, California, June 1983.

Gevarter W. B. Expert Systems: Limited But Powerful. *IEEE Spectrum*, Vol. 71, no. 8, p. 39-45, 1983.

Ginsberg M. L. Non-Monotonic Reasoning Using Dempster's Rule. In *Proceedings of the National Conference on Artificial Intelligence*, 1984, University of Texas, Dallas, Texas.

Goltzman D. Infrared Sensors for Air-to-Air FCS Applications (U), Hughes Aircraft Company, Ref. 23-30-00/107, August 1984.

Hayes-Roth F., Waterman D. A., Lenat D. B. (Eds.). *Building Expert Systems*, Addison-Wesley, Reading, Massachusetts, 1983.

Kinnucan P. Superfighters. *High Technology*, Vol. 4, No. 4, pp. 36-41 and 44-49.

Klahr P., McArthur D., and Narain S. SWIRL: An Object-Oriented Air Battle Simulator. In *Proceedings of the National Conference on Artificial Intelligence*, 1982, Carnegie-Mellon University, University of Pittsburgh, Pittsburgh, Pennsylvania

Lee C. A. Manual on Improved ARF, Hughes Aircraft Company, El Segundo, California, 1985 (in print).

Lenat D. B., Clarkson A., and Kiremidjian G. An Expert Sysem for Indications and Warning Analysis. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, Karlsruhe, West Germany

Lowrance J. D., Garvey T. D. Evidencial Reasoning: An Implementation for Multisensor Integration. Technical Note 307, SRI International, Menlo Park, California, December 1983.

Minsky, M. A Framework for Representing Knowledge. In P. Winston, ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, New York, 1975.

Pearl J. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. UCLA-ENG-CSL-8250, UC Los Angeles, California, June 1982.

Rauch H. E. Probability Concepts For An Expert System Used For Data Fusion. *The AI Magazine*, Fall 1984, pp. 55-60.

Rieger C. and Small S. Word Expert Parsing. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, 1979, Tokyo, Japan.

Riesbeck C. and Schank R. Comprehension by Computer: Expectation-based Analysis of Sentences in Context. Research Report 78, Yale University, 1976.

Schmolze J. G. and Lipkis T. A. Classification in the KL-ONE Knowledge Representation System. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, Karlsruhe, West Germany.

Shafer G. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey, 1976.

Shortliffe E. H. *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York, New York, 1976.

Spain D. S. Application of Artificial Intelligence to Tactical Situation Assessment. *IEEE*, pp. 457-464, May 31, 1983.

Stimson G. W. *An Introduction to Airborne Radar*, Hughes Aircraft Company, El Segundo, California, 1983.

Strat T. M. Continuous Belief Functions for Evidential Reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, 1984, University of Texas, Dallas, Texas.

Taylor J. W. *Jane's All The World's Aircraft: 1984-1985*, Jane's Publishing Co. Ltd., London, England, 1985.

Waltz E. L. Computational Considerations for Fusion in Target Identification Systems. Bendix Communications Disvision, Aerospace Systems Operation, Ann Arbor, Michigan, 1981.

Note: references 1-9 are from *Aviation Week and Space Technology*. Most articles did not have authors listed.

[1] March 12, 1984. pp. 136-173

[2] March 16, 1981. USAF Pushes Production, Performance. pp. 48-55.

[3] March 16, 1981. Soviets Press Production, New Fighter Development. pp. 56-61.

[4] August 8, 1977. Strength Sought at Least Cost. pp. 36-47.

[5] January 21, 1980. Soviets Press Antijamming Training. pp. 95-99.

[6] October 5, 1981. Soviet Union Defensive Buildup Detailed by Weinberger. pp. 18-22.

[7] August 31, 1981. Libyan Incident Spurs Deployment Shift. pp. 19-20.

[8] March 26, 1979. Soviets to Field Three New Fighters in Aviation Modernization Drive. pp. 14-16.

[9] October 20, 1980. Dornier Planning New STOL Flying Boat. pp. 65-66.

[10] Three New Soviet Air-to-Air Missiles in Service. *International Defense Review*, March 1978, p.400.

[11] New Soviet Aircraft: The RAMs. *Marine Corps Gazette, May 1983, pp. 20-22.*

[12] Special Report: Soviet ESM Capabilities Detailed. EW/DE, December 1978, pp. 60-68.

[13] Discussions with Mike Tom of Hughes Aircraft Company, October 1984.

[14] Discussions with Tracy Wickman of Hughes Aircraft Company, October 1984.