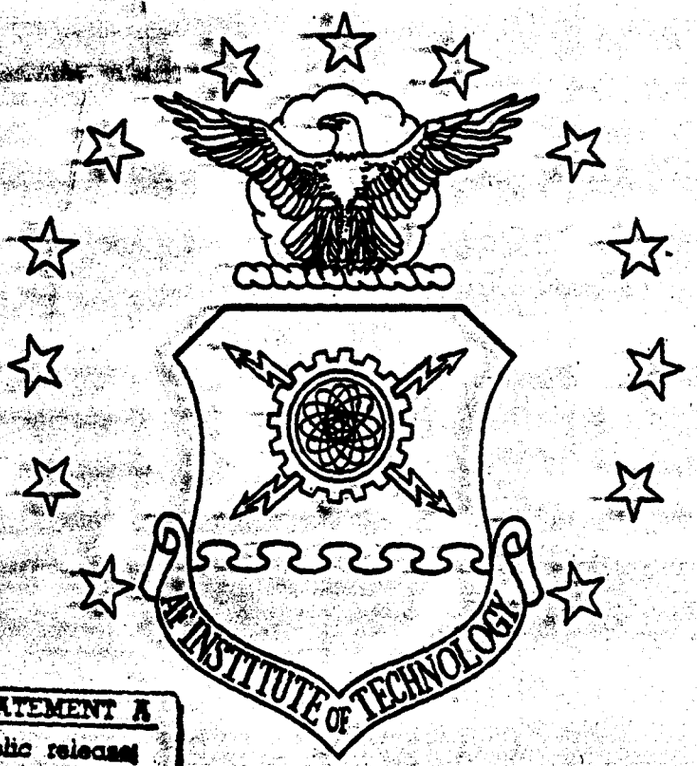

The Automated Wingman: A Computer Generated Companion for Users of DIS Compatible Flight Simulators

THESIS

Mark M. Edwards
Captain, USAF

AFIT/GCE/ENG/95D-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/95D-01

The Automated Wingman:
A Computer Generated Companion for
Users of DIS Compatible Flight Simulators

THESIS

Mark M. Edwards
Captain, USAF

AFIT/GCE/ENG/95D-01

"The views expressed in this thesis are those of the author
and do not reflect the official policy or position of the
Department of Defense or the U. S. Government"

AFIT/GCE/ENG/95D-01

THE AUTOMATED WINGMAN:

A COMPUTER GENERATED COMPANION FOR USERS OF DIS COMPATIBLE FLIGHT SIMULATORS

THESIS

Presented to the faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Mark M. Edwards, B.S.E.E, M.S.I.E.

Captain, USAF

December, 1995

Acknowledgments

The last 18 months have been a grueling but rewarding time. Who could pass up the tremendous opportunity to do pure research, earn a Masters Degree, and serve this great nation all at the same time? Certainly not me. This document is the pinnacle of my AFIT assignment and represents countless hours hard work. I believe that it is a product that I can be proud of, and I am proud of it. However, no effort of this magnitude is ever accomplished alone and I would be remiss if I did not thank those who have helped me along the way. Therefore, I take this opportunity to express my appreciation to all who have made this thesis possible.

First, I thank my classmates in the GCE, GCS and GE programs, especially Captains Lynda Myers, Neal Schneider, and Dave Kyger, whose help and camaraderie made the AFIT program enjoyable. To my professors, to numerous to list, who gave me the tools and knowledge required to accomplish this task, I say thank-you. To my committee, Dr Eugene Santos and Major Kieth Shomper, and my advisor, Lt Col Martin Stytz, in particular, thanks -- I could not have done without you. Thank-you all.

Most important of all, I thank my wife, Cindy, and my two boys, Stephen and Peter. Their love and understanding through the tough times has enabled me to persevere. Over the past 9 years I have dragged them three-quarters the way across the country, pulling them from comfortable and familiar surroundings and making them start all over again somewhere else. Through it all they have stood by and supported me. As a result, I owe them a debt that no amount of thanks can ever repay. Fortunately, as long as love is the currency, the balance sheet need never read "Paid in Full". I love you all!

Table of Contents

List of Figures

List of Tables

Abstract

A major problem encountered by users of distributed virtual environments is the lack of simulators available to populate these environments. This problem is usually remedied by using computer generated entities. Unfortunately, these entities often lack adequate human behavior and are readily identified as non-human. This violates the realism premise of distributed virtual reality and is a major problem, especially in training situations. This thesis addresses the problem by presenting a computer generated entity called the Automated Wingman. The Automated Wingman is a semi-automated computer generated aircraft simulator that operates under the control of a designated lead simulator. and integrates distributed virtual environments with intelligence. Access to distributed virtual environments is provided through the DIS protocol suite while human behavior is obtained through the use of a fuzzy expert system and a voice interface. The fuzzy expert system is designed around a hierarchy of knowledgebases. Each of these knowledgebases contains a set of fuzzy logic based linguistic variables that control the actions of the Automated Wingman. The voice interface allows the pilot of the lead simulator to direct the activity of the Automated Wingman. This thesis describes the design of the Automated Wingman and presents the current status of its implementation.

THE AUTOMATED WINGMAN:
A COMPUTER GENERATED COMPANION FOR USERS OF DIS COMPATIBLE FLIGHT SIMULATORS

1. Introduction

The mission of the Armed Forces of the United States is to prepare for war. This sole mission underlies the doctrine of each of the four branches of the United States military, including the United States Air Force (USAF). Thus, the basic doctrine of the USAF states, " Training should be as realistic as possible . . . . Exercises must replicate to the extent possible the chaos, stress, intensity, tempo, unpredictability, and violence of war" [USAF92]. The basic doctrine of the USAF also calls for joint training with other branches of the military as well as with allied nations. Unfortunately, factors in today's crowded world often work against these goals, making them difficult to attain through real life military exercises. Therefore, the defense community is looking to virtual environments and computer simulation to satisfy future training

1.1 Background

One of the first major thrusts in the use of synthetic environments for training purposes was the Defense Advanced Research Projects Agency (DARPA) sponsored SimNet Distributed Virtual Environment project [THOR88]. SimNet used several networked workstations to form a single distributed environment in which each node maintains a fully self-contained model of the environment. Participants, or actors, in this distributed environment broadcast their state either periodically or whenever their state has changed significantly from the last update. Each node then monitors the network for

the state of the actors and portrays this information within its own local environment. Designed primarily for ground-based combat, SimNet did not readily scale to high performance aircraft simulators. However, SimNet was successful in showing that virtual environments could be effectively used as training tools and research has continued in an attempt exploit this capability [THOR88].

A real combat environment includes aircraft, helicopters, and other airborne threats as well as soldiers, tanks, and other ground-based entities. Airborne vehicle simulators move faster and turn quicker then their ground-based counterparts, requiring frequent updates of position, velocities, and orientation. Therefore, in 1989, DARPA, now known as the Advanced Research Projects Agency (ARPA), commissioned a project to develop the Distributed Interactive Simulation (DIS) standard, based on the work of the SimNet project, that incorporated advances in network communication and simulator technology. True to its SimNet origins, each DIS compatible simulator maintains its own models, terrain, and entities representing external simulators. The DIS standard is currently at version 2.0 and several research projects are underway to add more capability and flexibility. DIS continues to evolve and improve as more and more emphasis is placed on training through virtual reality.

To address the need for high performance airborne simulators, DARPA also sponsored AFIT to develop a low cost, DIS-compliant flight simulator [MCCA94]. This project, known as the AFIT Virtual Cockpit (VC), has been the subject of several AFIT master's theses [SWIT92] [ERIC93] [MCCA94]. The VC uses commercially available off-the-shelf hardware and software to create an immersive environment based on the

cockpit of an F-15E. The immersion is achieved using a variety of head-mounted displays (HMDs) and a hands-on-throttle-and-stick arrangement using a Thrustmaster joystick and throttle. A working version of the Virtual Cockpit was successfully included, on a limited basis, in a series of simulation exercises called Zealous Pursuit [MCCA94]. Work continues on the Virtual Cockpit, adding new features and improving old ones.

DIS is not simply a research project. It is in use today, providing both a realistic training environment for military units and a research test-bed rich in potential. Using a dedicated wide-area network called the Defense Simulation Internet (DSI), simulators at training centers around the country interact on a regular basis in limited training exercises. Most facilities conducting DIS research, including AFIT, are also connected to the DSI. Since DIS is designed to accommodate multiple simultaneous exercise simulations, training and research can be performed at the same time. Researchers can also be included in exercises using observation platforms such as AFIT's Synthetic BattleBridge [HADD93]. This allows researchers to get a feel for how DIS is performing, what its limitations are, and what areas need improvement. This has led to the identification of several issues that must be addressed before DIS can be used for large-scale training exercises.

1.2 Problem Statement

One of the problems in current DIS exercises is that the number of aircraft simulators present does not adequately reflect the number of aircraft that would actually participate in a real exercise. It is completely unrealistic that a major European conflict

will be fought with only four combat aircraft, a typical number in a distributed simulation. Because of the limited number of simulators available, aircraft sorties are usually flown as single-ship formations. This is unrealistic and goes against the principle of realistic training spelled out in USAF doctrine. In an actual exercise, these single-ship formations would consist of at least two aircraft, if not more. This lack of realism degrades the benefit derived from the simulation and needs to be corrected.

1.3 Research Objective

I propose to develop an architecture that will provide users of a variety of DIS compatible aircraft simulators with one or more unmanned but intelligent wingmen. These wingmen will use pre-mission planning, situational awareness, and voice control as inputs to an expert system equipped to deal with uncertainty and approximation. Fuzzy inferencing techniques and linguistic variables provide a close approximation to human reasoning and will be used provide this capability to the expert system. Mission planning will provide the wingman with the overall goal of the mission and a rough framework in which the mission is to be accomplished. Linguistic variables and approximate reasoning techniques to will be used to evaluate the current situation and maneuver within that framework. Once the appropriate action has been determined, the Automated Wingman will execute the action using standard USAF fighter tactics, or a close approximation of them, to fly and fight alongside the flight leader in the manned simulator.

1.4 Scope

The capabilities of the Automated Wingman will be limited in that its actions will be tied to those of the flight leader. The Automated Wingman will not act as an

independent force capable performing its mission without human direction. Instead, it will perform the appropriate maneuvers based on the situation and verbal commands from the flight leader. Unlike in the real situation, the command set available to the flight leader will be limited to a small number of predetermined orders and linguistic variables. The Automated Wingman will have the capability to understand several maneuvers, such as aileron loop and figure-8 as well as a few concepts such as "start bomb-run" or "evade SAM". However, the pilot will not be able to teach tactics or lay out a plan to the Automated Wingman as a real flight commander might do. Also, the first version of the Automated Wingman will not be able to complete the mission on it's own. If the leader is shot down then the mission would have to be aborted. Therefore, the ability of the pilot to swap cockpits will be a key feature. As long as the pilot has a plane to fly, other Automated Wingmen will continue to apply their fuzzy rules and determine the appropriate action for the situation.

2. Background

2.1 Introduction

The Automated Wingman builds upon several key areas of research. In order to understand the design of the Automated Wingman and the significance of its results, these topics must be familiar to and understood by the reader. The purpose of this chapter is to identify and introduce these important subjects. The topics covered in this chapter are Distributed Interactive Simulation, Fuzzy Logic, Expert Systems, and another research project called TacAir-Soar. This chapter is not intended as a tutorial in these topics. Instead, this chapter will introduce the subject matter, discuss the relevance of the subject to the Automated Wingman, and refer the reader to appropriate material should further study be required. Once armed with an understanding of the background material, the reader will be ready to proceed with the design and implementation of the Automated Wingman.

2.2 Distributed Interactive Simulation

The Distributed Interactive Simulation (DIS) protocol suite is a major part of the Automated Wingman, It not only provides the rationale for development of the Automated Wingman, it also provides the means for the Automated Wingman to be seen by other players in a distributed simulation. This section introduces DIS and an implementation of an interface to the DIS protocols developed at AFIT that makes DIS

easy to integrate into simulators and other programs intended for use in distributed simulations.

Distributed simulations are conducted by linking computer-based combat simulators together over a common network. Each simulator controls one or more entities (tanks, planes, etc.) that move within a common environment (terrain, weather, lighting, etc.). As the entities move, each simulator keeps all the other networked simulators informed of the location and status of it's local entity or entities. Representations of remote entities are presented to the users of each simulator. In this fashion, entities can interact (acquire, track, shoot, and destroy) with other entities, local or remote, within the simulation using a predetermined protocol suite. The DIS protocols are an example of such a suite that enables networked simulators to interact within a distributed environment.

The DIS protocol suite is the key technology enabling distributed simulations that can include thousands of entities. Distributed simulation research has been extensively documented and the interested reader is referred other sources for detailed information on this subject. [GOSS94] is a distributed simulation tutorial and a good place to start for those new to the subject. Several excellent summaries of DIS exist including [BLAU94] and [STYT95]. [IST94] contains a glossary of terms used in DIS. The standards document that fully enumerates the DIS protocol is [IEEE93]. Annotated bibliographies of other references may be found on the World-Wide Web at http:/www.afit.af.mil/Schools/EN/ENG/LABS/GRAPHICS/annobibs/annobibs.dis.html.

The key feature of the DIS protocols is that each simulator is required to maintain knowledge of all entities within the simulation. Further, when an entity broadcasts it's position over the network it is also required to broadcast parameters that will allow each simulator to calculate that entity's changes in position and velocity over time. Parameters are broadcast using packets of data called Protocol Data Units (PDUs). Positions are then calculated using the data in these PDUs and a set of dead-reckoning algorithms provided by DIS. Each entity is also required to dead-reckon it's own position. When the entity's actual position differs from the dead-reckoned position by a pre-determined threshold the entity broadcasts it new position and dead-reckoning parameters. While introducing some positional error, this method reduces the network traffic and enables large-scale distributed simulations with thousands of participating entities.

AFIT has been involved with DIS research since 1992 and has developed several DIS-compatible simulators [SWIT92], [ERIC93], [GERH93], [MCCA94], [DIAZ94], [KUNZ94], [VAND94], viewing platforms [WILS93], [SOLT93], [HADD93] [STYT94] and an activity recorder analogous to a VCR [FORT94]. In order to facilitate the development of these projects a DIS interface, called Object Manager, was developed [SHEA92]. The Object Manager package consists of two distinct parts, the Entity Object Manager and the Application Object Manager. The first of these, the Entity Object Manager, keeps track of all the entities in the simulation. It performs all dead-reckoning and maintains the list of active entities. When an application requires information about the state of the objects in the simulation it accesses the Entity Object Manager for this information. The Application Object Manager, on the other hand, puts the applications

entity on the network. Once again, it handles the required dead-reckoning and determines when the data must be broadcast to the rest of the simulators. The application keeps the Application Object Manager informed of it's movements through a procedure specifically designed for that purpose. The Object Manager package removes the responsibility of the DIS interface from the application developer and is an important part of the Automated Wingman.

2.3 Fuzzy Logic

Although the Automated Wingman is not designed as an independent, intelligent agent, it still must posses the capability to reason about it's environment and take appropriate actions. To accomplish this, the Automated Wingman uses an approximate reasoning technique called fuzzy logic. Fuzzy logic provides a method for representing knowledge and dealing with the ambiguity and uncertainty inherent in that knowledge. Traditional reasoning techniques, such as modus ponens [WINS93], can be extended to deal with knowledge in this form. The result is a conclusion that takes uncertainty and ambiguity into account and more closely resembles the decision that would have been made by a human under the same circumstances [KOSK93]. For this reason, the Automated Wingman uses fuzzy logic to help make it indistinguishable from human controlled entities within a simulation.

A complete overview of fuzzy logic is beyond the scope of this document. However, like DIS, fuzzy logic and it's applications have been extensively documented and published. The original paper on fuzzy logic is [ZADE65], which provides a clear description of fuzzy logic in the terminology of discrete mathematics. Other excellent

introductions include [KOSK93], [MUNA94], and [COX92]. Detailed information about fuzzy operators and fuzzy set theory is in [ZADE92], [ZIMM87a], [DUBO80], [KLIR95], and [KOSK92a]. Annotated bibliographies of many of these papers can accessed on the World-Wide Web at http:/www.afit.af.mil/Schools/EN/ENG/LABS/GRAPHICS/ annobibs/annobibs.fuzzy.html. Some of these papers describe objects known as linguistic variables, which are important to the Automated Wingman

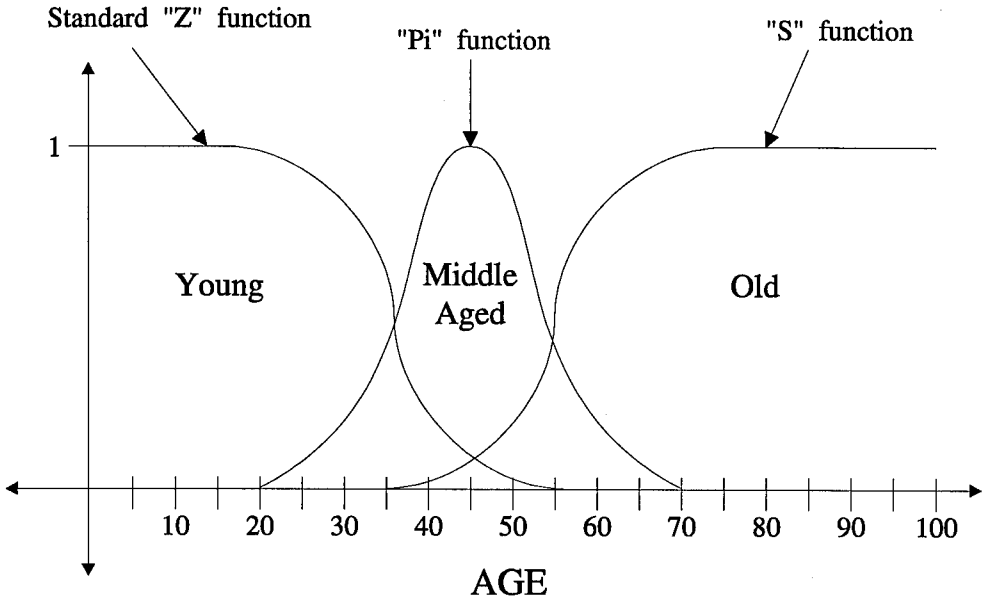

Figure 2-1 The Linguistic Variable AGE

The concept within fuzzy logic that is most applicable to the Automated Wingman is that of a linguistic variable. A linguistic variable, such as temperature, describes a quantity or an idea that is best represented by fuzzy sets, such as hot, warm, and cold. These are called term sets. The linguistic variable can be set to one of these term sets or, by "fuzzifying" a crisp value and determining to which term set the crisp value belongs, the linguistic variable can be set to a fuzzy set that encompasses several of the term sets. The linguistic variable then takes on the value of the term sets that apply, not the crisp

value itself [ZADE75] [ZADE79] [SCHW90]. For example, the concept of age is often described in terms of young, middle-aged, and old. Figure 2.1 shows one possible design for the linguistic variable age. In this example, a person who is 15 years old is "young" while 75 year-old person is "old". A 45 year-old person could be considered "young" or "old" to a very small degree but is primarily called "middle-aged" [ZADE75]. Fig 2-1 also shows the standard "Z", "Pi", and "S" membership functions that are common in linguistic variable design. These functions embody the ambiguity inherent in the concept of age and give the linguistic variable its descriptive power. The Automated Wingman will capitalize on this descriptive power to create an expert system capable of flying within a distributed environment.

2.4 Expert Systems

An expert system is a program that combines knowledge about a particular domain with reasoning mechanisms for the purpose of making deductions about the domain from some known facts. An expert system shell, on the other hand, is a program that contains only the reasoning mechanisms. The knowledge about the domain must be obtained from domain experts through a process called knowledge acquisition [GIAR94] and reduced into a form readable by the expert system shell. The reduced form is called a knowledgebase. The use of expert system shells allows expert systems to be developed for a wide variety of domains without having to develop specialized reasoning software for each specific domain.

The heart of the Automated Wingman is a fuzzy expert system shell called FuzzyCLIPS. CLIPS, which stands for C Language Integrated Production System

(CLIPS) is an expert system shell developed by NASA [JSC92]. The term "Production System" indicates that CLIPS represents knowledge as a series of rules of the form "if A is true then conclude that B is true" [WINS92]. The Knowledge Systems Laboratory of the Institute for Information Technology, National Research Council Canada has written extensions to CLIPS to handle fuzzy logic and the resulting program is called FuzzyCLIPS. Both CLIPS and FuzzyCLIPS are written in the C programming language and provide a C language interface to CLIPS functionality. Both can run either as a stand-alone application, a stand-alone application augmented by user developed C code, or as an embedded expert system shell within a user application. The Automated Wingman runs FuzzyCLIPS in the last of the three modes -- as an embedded expert system.

Expert systems have been under development since the early 1950s under the umbrella of artificial intelligence. Hence, a great deal of literature is available to describe the different types of systems, such as case-based reasoning systems and production systems, as well as how they operate. A good introductory text is [WINS92], particularly for solid foundation in knowledge representation, production rules, and chaining. [GIAR94] explores expert system development in general and shows examples using CLIPS (the author is one of the developers of CLIPS). This book also contains a section on reasoning under uncertainty using fuzzy logic. The FuzzyCLIPS software and manual [KSL94] are available free of charge on the World-Wide Web at http:/ai.iit.nrc.ca/fuzzy.html. Annotated bibliographies of other articles may be found at http:/www.afit.af.mil/Schools/EN/ENG/LABS/GRAPHICS/annobibs/annobibs.html.

2.5 *TacAir-Soar*

The TacAir-Soar project is the first large program to undertake the task of simulating pilot behaviors for use in DIS environments [LAIR95]. It's purpose is to develop independent computer generated forces, called IFORs, that can execute a mission without human intervention. To do this, TacAir-Soar builds upon an architecture for general intelligence called Soar that has been under development since 1983 [ROSE93]. The relationship between TacAir-Soar and Soar itself is intricate and cannot be explained without an understanding of the Soar architecture. Therefore, this section will describe Soar in enough detail to serve as a foundation for understanding TacAir-Soar. Ultimately, TacAir-Soar will be described and related to the goals of the Automated Wingman.

The goal of the Soar program is to identify the key elements of general intelligence and then design a software architecture that implements these elements. The design of Soar is based on the concept of a *universal weak method* [ROSE93]. A universal weak method attempts to reason and solve problems by applying general, domain-independent structures and knowledge under the assumption that there is one approach that can solve all problems. Using a universal weak method, Soar attempts to exhibit general intelligence by performing tasks at the same level of proficiency that a human can. This includes reasoning and learning. Since modeling human intelligence is key to the success of the Soar project, Soar is based upon a cognitive model of human intelligence [ROSE91].

The model of human intelligence that Soar is based on separates human intelligence into three layers. At the bottom is the neural band. The neural band encompasses symbolic knowledge that is accessed directly without deliberate thought. This includes reflexes, autonomic functions such as breathing and balance, and the million other things we do everyday without even thinking about them. The next level is the cognitive band. The cognitive level is where most human knowledge is stored, accessed, and used for deliberation and decision making. Goal attainment is measured in the cognitive level and operators are composed to achieve these goals as well. Finally, the last level is the rational band. This band is goal-oriented, knowledge-based, and strongly-adaptive. It provides humans with higher order rationalization functions such why an action is being performed or what goal is to be achieved. These bands form an architecture for human intelligence that is amenable to implementation in computer software [ROSE91].

Of the three levels described above, Soar implements the cognitive band. This is because there is limited intelligence embodied in the neural band and the rational band is more a mechanism for understanding what we do, not why we do it. Soar further divides the cognitive band into 3 sub-levels. The lowest of these levels is the memory level where symbolic knowledge is stored and can be accessed. The primary data structure is called a production and uses object-attribute pairs to represent knowledge. Productions are analogous to if-then rules in that they have a condition-action structure. In order to activate, or "fire", the action, all the conditions of a rule must be satisfied. In contrast to regular production systems which use conflict resolution to determine which rules to fire,

Soar fires all productions that apply and lets the next level decide upon the appropriate action. In order to aid the next level in making this choice, Soar also has a special memory data structure called a preference. Preferences encode knowledge about which actions are best for a given situation or set of conditions. Once all the applicable productions and preferences have fired the next level takes over [ROSE91].

The next level is called the decision level. It uses the productions and preferences from the memory level to make elementary deliberate decisions. If the production and preferences all point to one action then the decision is trivially easy. However, since there is nothing to prevent productions and preferences from conflicting, there is a possibility that an impasse will occur. Breaking the impasse requires knowledge about the goal that the system is attempting to achieve. This is the domain of the next level [ROSE91].

The top level is the goal level. The responsibility of this level is to determine what to do when the decision level reaches an impasse. Usually this level will attempt to break the impasse. According to its universal weak method, Soar will attempt this by creating a "problem space" and a sub-goal that will add enough knowledge to break the impasse. Soar then works on this sub-goal until it is either solved or results in another impasse. This process recurses to the level necessary to bring enough information to bear to solve the problem. This is very similar to the problem reduction method [WINS92] but with an important difference. Throughout this process the entire goal stack is available for inspection by all problem spaces. If knowledge uncovered in a sub-goal solves a super-ordinate problem then the super-ordinate problem space is closed and all

sub-problems are deleted because they are now irrelevant. This opportunistic approach to problem solving enables Soar to reach solutions faster than depth-first search oriented architectures [ROSE91].

Another feature of Soar is that it has the ability to learn through a process known as chunking. As previously described, Soar resolves impasses at the decision level by breaking the problem into sub-problems and solving the sub-problems. Once the impasse has been broken, Soar saves the conditions that caused the impasse and the operators that were used to resolve it. These new "chunks" of knowledge are stored away for retrieval later should the same situation arise again. Thus, an impasse can sometimes be avoided. These chunks not only solve problems already encountered, but can be sufficiently general to solve problems that are similar, but not identical, to the original problem. This is a powerful capability that supports the Soar goal of achieving general intelligence [ROSE91].

The Soar project has been under development since 1982 and has already been used to demonstrate general intelligence. R1-Soar was a project to demonstrate automated computer configuration similar to X1/RCON [ROSE93]. Other domains in which Soar has been successfully applied include algorithm and software design, medical diagnosis, blood banking and factory scheduling [ROSE93]. However, the project that is by far the most relevant to the Automated Wingman is TacAir-Soar. TacAir-Soar has demonstrated the ability to produce many pilot behaviors and has been used successfully in a distributed simulation.

TacAir-Soar builds upon Soar and another architecture called ModSAF. Soar provides the intelligence and ModSAF provides the aircraft dynamics and access to distributed simulations. Since Soar's intelligence is general in nature, the developers of TacAir-Soar have focused on providing Soar with the knowledge specific to flying an airplane and fighter tactics. As of early 1995, TacAir-Soar consists of approximately 200 operators spread over 24 problem spaces and about 1700 productions [TAMB95]. The capabilities provided by this knowledgebase are impressive, as was demonstrated in a distributed simulation in which TacAir-Soar was a participant [LAIR95].

TacAir-Soar was included in the Synthetic Theater of War - Europe (STOW-E) exercise held in November of 1994. The purpose of STOW-E was to demonstrate the feasibility of a large-scale distributed interactive simulation by integrating real forces, human-controlled simulators, and computer generated IFORs [ROGE94]. The exercise included more than 1800 entities over a 3 day period. Although there were problems, TacAir-Soar demonstrated that computer-generated air forces could participate in DIS environments. It was even successful in shooting down several manned simulators (although that was not the usual case) [LAIR95]. The successful inclusion of TacAir-Soar marked the first time that a general intelligence architecture such as Soar has participated in a large-scale distributed simulation.

Figure 2-2 shows a portion of the TacAir-Soar goal hierarchy. Mission parameters are entered into the program prior to beginning the simulation. While running, TacAir-Soar follows a top-level goal labeled "Execute-Mission". That goal is obviously too abstract to be implemented directly; no information exists in memory to

attain that goal and an impasse occurs. Using the problem spaces developed by the TacAir-Soar team, the system attempts to break the impasse by recursively dissecting the problem and examining available operators [TAMB95]. Productions are used to take known facts about the environment and suggest appropriate operators within the problem space. Preferences indicate which actions are more desirable than others [ROSE91]. As Soar moves further down its goal hierarchy the goal stack builds up, accumulating all the goals that are currently at an impasse. If Soar discovers a new piece of information, either as a result of applying an operator or from one of its aircraft sensors, it checks the goal stack to see if any of the blocked goals can be resolved. When Soar finds an operator that it can apply Soar executes it, and the corresponding problem space and all it's sub-spaces are "rolled-up". This process repeats until the "Execute-Mission" goal is achieved and the simulation is over [TAMB95]. An extended example of TacAir-Soar in operation can be found in [TAMB95].

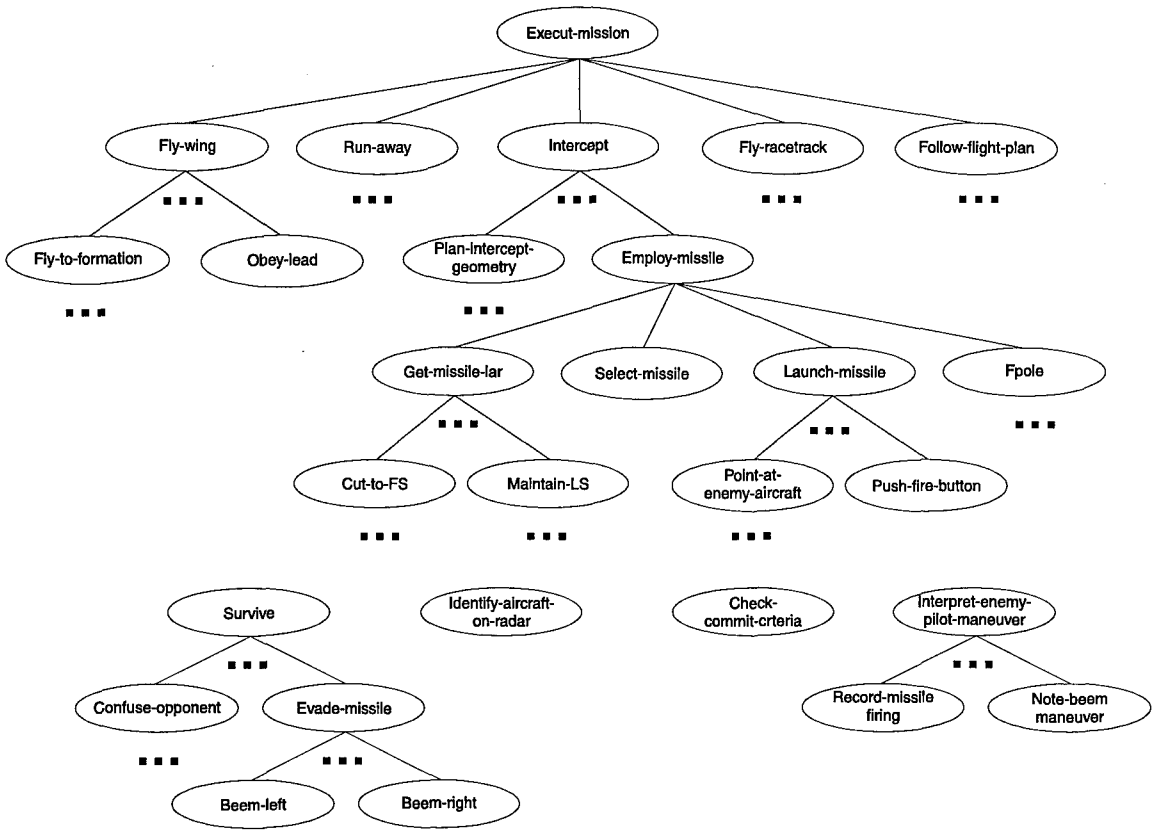

Figure 2-2 A Portion of the TacAir-Soar Goal Hierarchy

From the description of TacAir-Soar it is clear that what distinguishes TacAir-Soar from Soar itself is that Soar provides the capability for general intelligence while TacAir-Soar provides the knowledge necessary for pilot behaviors. This is analogous to the distinction between an expert system shell and the knowledgebase that provides the system with its expertise. This separation has allowed the developers of TacAir-Soar to concentrate on encoding pilot behaviors and tactics instead of the mechanisms of intelligence itself [TAMB95]. The result is impressive. However, human pilots are still able to defeat TacAir-Soar aircraft on a regular basis, as demonstrated in STOW-E. Therefore, something is still missing. Until computer generated forces can sustain an

even loss ratio against human pilots, there is a chance that the value of their participation may be compromised.

2.6 Conclusion

This chapter has examined several subject areas related to the Automated Wingman. These include the Distributed Interactive Simulation (DIS) protocols, fuzzy logic, and expert systems. Also, the TacAir-Soar project was presented. TacAir-Soar attempts to create an independent entity for distributed simulations using an architecture for general intelligence called Soar. TacAir-Soar has demonstrated the successful use of computer generated entities in a distributed simulation but still suffers from a high loss rate to human pilots. The Automated Wingman uses a different approach to generating semi-automated, as opposed to fully automated, entities. The Automated Wingman attempts to address TacAir-Soar's shortfall through the use of fuzzy logic.

3. Requirements

3.1 Introduction

To successfully fulfill its mission, the Automated Wingman must exhibit certain features and capabilities, including DIS compatibility, autonomous route planning and flight control, and responsiveness to enumerated voice commands. These capabilities are important because they maintain the illusion of human control. The features include machine independence and context switching. These make the Automated Wingman more flexible and easier to employ in a distributed environment. Each of these will be described in detail in this chapter.

3.2 DIS Compatibility

The Automated Wingman must be DIS-compatible in order to participate as an actor within a distributed exercise. While flying a mission, the Automated Wingman will be under the control of the lead aircraft simulator. However, all communication must occur using DIS PDU's. For this reason, a critical requirement of the Automated Wingman is that it be able to read DIS Entity-State PDU's and perform to dead-reckoning as specified by the protocol. At the same time, the Automated Wingman cannot increase the density of entities within a distributed simulation unless it is visible to all of the other entities. Hence, the Automated Wingman must also be able to broadcast DIS Entity-State PDU's that describe its own state, including the specified dead-reckoning parameters, for

the other entities to read. Therefore, the Automated Wingman must be able to read and broadcast, at a minimum, DIS Entity-State PDU's.

The DIS protocol suite contains another family of protocols relevant to the Automated Wingman. This family is the Radio Transmission PDUs. Using a capability described in section 3.4, the Automated Wingman must be able to accept and respond to verbal commands from the lead simulator. The Radio Transmission PDU family is the mechanism that supports this functionality. Therefore, the Automated Wingman must recognize and decode Radio Transmission PDUs in addition to Entity-State PDUs.

3.3 Autonomous Route Planning and Flight Control

Like a real pilot, the Automated Wingman must be capable of flying it's own plane. It must be able to maintain straight and level flight as well as perform basic directional maneuvers such as climb, dive, and bank to turn. Given its current location, orientation, and velocity, the Automated Wingman must be able to use those basic directional maneuvers to fly to a specified point in the environment with no external intervention. Throughout these maneuvers the Automated Wingman must also manage it's own airspeed. For example, if the Automated Wingman is ordered to fly in echelon formation with the lead, then it must determine the coordinates of the echelon formation position, use the basic maneuvers to fly to this position (note that the desired position is a moving target), and control its airspeed to avoid overshooting the desired position. This is a complex requirement but absolutely necessary for semi-autonomous behavior.

Many flight maneuvers are combinations of the basic directional maneuvers. In order to behave like a human pilot, the Automated Wingman must know how to employ

these more complicated maneuvers. For example, if the Automated Wingman arrives at the desired coordinates and orientation early, it must perform some delaying maneuver. As "S" turn will do this nicely. However, an "S" turn is a relatively complicated maneuver that requires several basic bank turns while maintaining altitude. This task is far more complicated than just a simple turn. Therefore, the Automated Wingman must also be able to plan a series of route points in order to execute more complicated maneuvers.

Finally, the Automated Wingman must also know when to employ these maneuvers. This means it must have a goal it is trying to achieve and a planning system to guide its actions towards that achievement. These goals should be hierarchical in nature, i.e., a set of sub-goals will be employed to achieve a super-ordinate goal. Using the goal structure, the Automated Wingman could then employ different tactics and maneuvers in an effort to achieve each goal. In this way, the Automated Wingman will know when to employ an "S" turn, or a barrel roll, etc. Therefore, route planning is an essential element for the Automated Wingman.

3.4 Voice Commands

The leader of a formation of real aircraft is usually able to communicate, either verbally or visually, with his wingman. Visual communication, other than follow-the-leader, is not an option for the Automated Wingman. Therefore, verbal communication becomes more important. The Automated Wingman must be able to receive, interpret and execute a limited set of voice commands from the lead simulator as if it were receiving radio signals. These commands include basic maneuver commands, commands to employ tactics, target designation, flight mode (formation versus independent), etc. Additionally,

the Automated Wingman must be able to distinguish between its lead and other voice traffic through the use of call signs. This is so that multiple wingmen do not interpret commands meant for other instances of the Automated Wingman. The ability to receive and respond to voice commands enhances the realism of the Automated Wingman and increase the training value of a distributed simulation.

3.5 Machine Independence

Most simulators require a high-end workstation to render graphical representations of the distributed environment. Only certain computers, such as the Silicon Graphics, Inc. (SGI) Onyx Reality Engine, have the specialized graphics hardware necessary for rendering graphics acceptable for training purposes. However, the Automated Wingman is not required to render any graphics. Hence, it does not require a high-end workstation to operate on. Therefore, the Automated Wingman cannot use any SGI specific software libraries, such as the Performer graphics libraries. The one exception is the Object Manager, which is specific to the SGI architecture because of its network interface. Even then, a clearly defined interface must exist between the Automated Wingman and the Object Manager package so that the Object Manager can be easily replaced if the Automated Wingman is ported to another computer architecture. This machine independence will increase the value of the Automated Wingman since it can run on cheaper computers, freeing the expensive ones for human trainees.

3.6 Context Switching

Since the purpose behind this project is realistic training, the Automated Wingman is also designed to allow the pilot of the manned lead simulator to swap cockpits with the Automated Wingman, maximizing the training opportunity of the pilot. This will involve determining the appropriate interface and DIS protocol PDU usage in order to accommodate this feature. Although this capability will initially be provided through the AFIT Virtual Cockpit, the interface will be modular and fully documented so that other simulators can be modified to make use of it. Cockpit swapping is the topic of another thesis [SCHN95] and will not be described further in this document.

3.7 Conclusion

The Automated Wingman must be DIS-compatible and be able to fly its own airplane. Further, it must be able to receive and implement voice commands from the lead simulator. Both the voice commands and the nature of flying require that the Automated Wingman have a hierarchical goal structure and planning capability, in order to satisfy its mission requirements. Through these features, the Automated Wingman will meet its goal of increasing the number of aircraft involved in training simulations while maximizing the opportunity for trainees to experience realistic combat situations.

4. Design Decisions

4.1 Introduction

The Automated Wingman is a complex system that embodies artificial intelligence as well as traditional programming. In order to accommodate both of these facets, I designed an architecture that separates them into two distinct modules with a clearly defined interface. The airplane module provides the aerodynamics, weapons systems, and flight parameters such as location, orientation, airspeed, etc. The FuzzyPilot provides the system with the reasoning and decision making facility required for human-like behavior by interfacing with a hierarchy of fuzzy knowledgebases. The basic architecture ties these two modules together through a common interface in order for them to work together. It also provides DIS access through the Object Manager package (see section 2.2). This chapter describes the design of the basic architecture, the aircraft module, the FuzzyPilot module, the knowledgebase hierarchy, and other parts of the Automated Wingman that enable it to fly within a distributed simulation.

4.2 Overall System Design

This section describes the top-level design of the Automated Wingman. I identify the inputs and outputs necessary for the Automated Wingman to satisfy the requirements laid out in Chapter 3. I then decompose the Automated Wingman into a set high level modules that form the basis of the remainder of the design. This section lays the groundwork for the design outlined in the rest of the chapter.

4.2.1 Context Diagram

The first order of business in any design is to identify the system inputs and outputs. This is usually done with a context diagram, which is shown in Figure 4-1 for the Automated Wingman. The inputs are enumerated voice commands from the lead simulator, terrain data, information on other entities, and any kind of electromagnetic emission data from within the simulation. The required outputs are the coordinates and dead-reckoning parameters of the Automated Wingman as well as any active weapons that may have been launched from the Automated Wingman. With the inputs and outputs identified, I proceeded to create a high level design of the Automated Wingman's architecture.

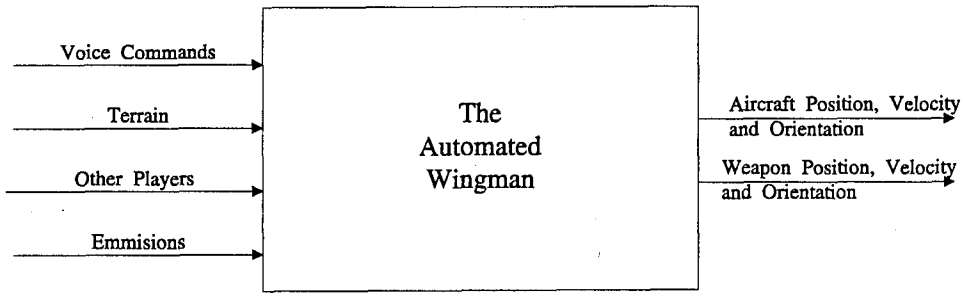

Figure 4-1 Automated Wingman Context Diagram

4.2.2 Module Diagram

Figure 4-2 shows the division of the Automated Wingman into modules for the different types of processing that are required. In developing this design, I attempted to mirror reality to the fullest extent possible. For example, a real pilot does not have to

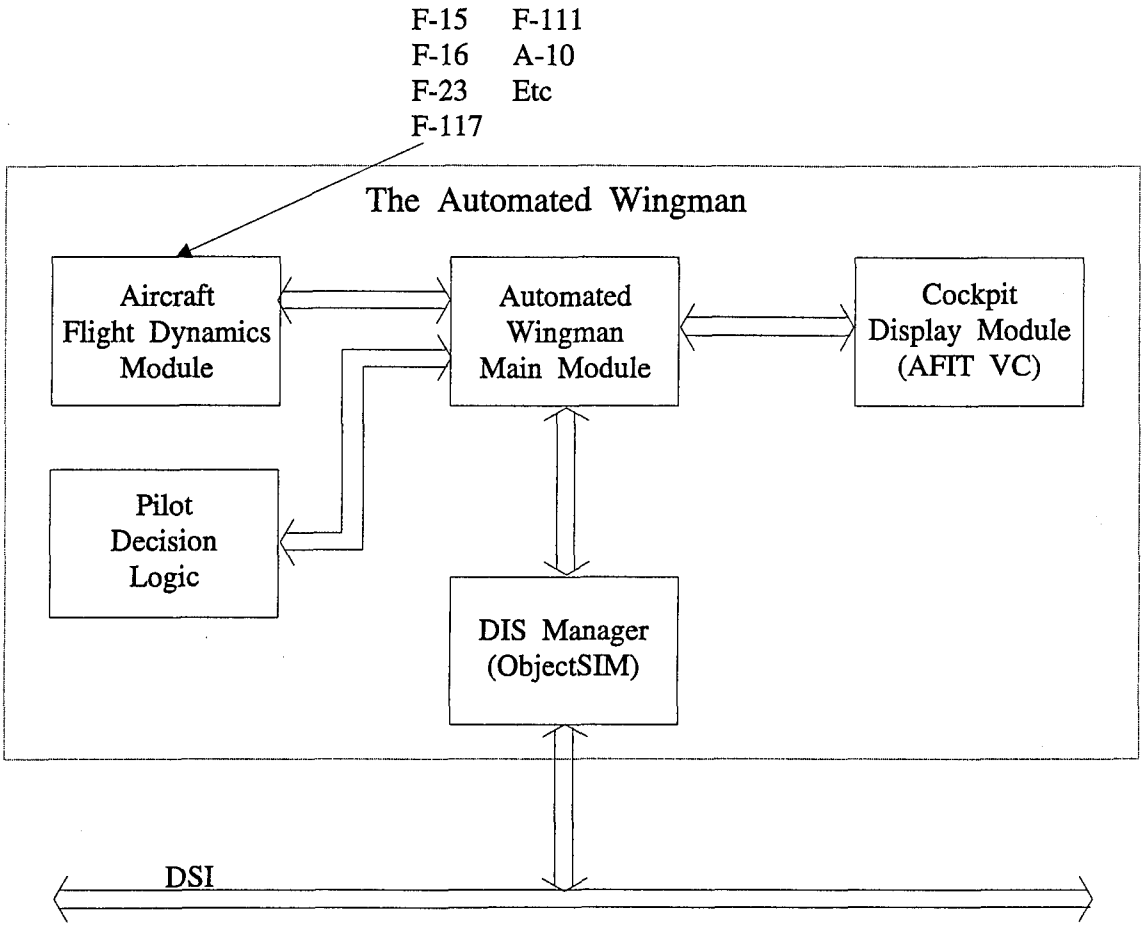

Figure 4-2 Automated Wingman Module Design

compute the airspeed of the aircraft from the velocity vector, it is simply displayed on the airspeed indicator. Using that analogy, I determined that the most appropriate design is to have a module that represents the pilot and another that represents the airplane. That philosophy is evident from Figure 4-2.

The Pilot Decision Logic module represents the pilot of the Automated Wingman. It embodies the fuzzy expert system that provides the Automated Wingman with the intelligence needed to fly the plane and perform route planning tasks. Information about the aircraft's current status, the lead simulator, and the environment are input to this module. It then uses a goal hierarchy and programmed knowledge to generate commands that the airplane can respond to. In short, the Pilot Decision Logic module is the brain of the Automate Wingman.

The Aircraft module provides all the features that an actual aircraft provides. It consists of an aerodynamics model, gauges, a weapons store, and other hardware found in an airplane. To provide control, the Aircraft module also contains a throttle and stick as well as switches for the afterburner and speedbrake. This module is designed to be expandable to provide more capabilities as the system matures.

The DIS module is also an important part of the Automated Wingman. It provides the Automated Wingman with the capability to receive and broadcast DIS PDU's. This module enables the Automated Wingman to meet the DIS requirements specified in chapter 3. A full description of the DIS module can be found in [SHEA92].

The Cockpit Display Module is intended as a future add-on to allow an observer to ride in the cockpit and view first-hand the actions of the Automated Wingman. Based on the design of the AFIT Virtual Cockpit, it provides an optional immersive environment. Since this capability is not a requirement for the Automated Wingman, the design of this module has not proceeded very far. However, it is included to demonstrate the flexibility of the underlying architecture.

4.3 Basic Architecture

The Automated Wingman is designed with flexibility and scalability in mind. The goal was to maintain a clear separation of tasks while developing an efficient system. To do this, I used an object-oriented modeling tool called Object Modeling Technique (OMT) [RUMB91] to create an object model and a dynamic model to show how the Automated Wingman functions. These are shown and explained in this section. I then extract the two central objects of the design, the Airplane object and the FuzzyPilot object, from the overall system model and describe them. These objects provide the fundamental foundation for the development of the Automated Wingman.

4.3.1 Object Model

The object model shows the system decomposed into individual units, called objects, that function together to produce the desired system behavior [RUMB91]. Connections between objects show relationships and how they interact. The object model of the Automated Wingman is shown in Figure 4-3. This diagram clearly shows that the Automated Wingman is associated with the two Object Manager classes, Entity_Object_Manager and Cockpit_Object_Manager. These objects manage the Automated Wingman's interaction with the DIS protocols. The data passed between the Automated Wingman and these two objects is shown as association objects. The Automated Wingman also consists of an Airplane object and a FuzzyPilot object, both of which are described in detail in sections 4.3.3 and. 4.3.4. However, the interaction of these two objects bears further discussion.

The interface between the Airplane and FuzzyPilot objects is also clearly defined in Figure 4-2. This interface is in the form of two association objects, one for gauges and one for controls, maintaining the analogy of a real pilot in an airplane. The Gauge object contains information such as coordinates, altitude, airspeed, climb rate and other information that one would expect to find in a cockpit. This object is used to pass information from the Airplane object to the FuzzyPilot object described in section 4.3.4. The Controls object keeps the current TAS (throttle, stick, rudder, afterburner and speedbrake) settings and passes this from the FuzzyPilot to the Airplane. An alternative scheme is to make both of these part of the Airplane class. However, that would require the FuzzyPilot to have explicit knowledge of the Airplane class. Using the association object maintains the separation between the Airplane and the FuzzyPilot and is therefore the cleaner design choice.

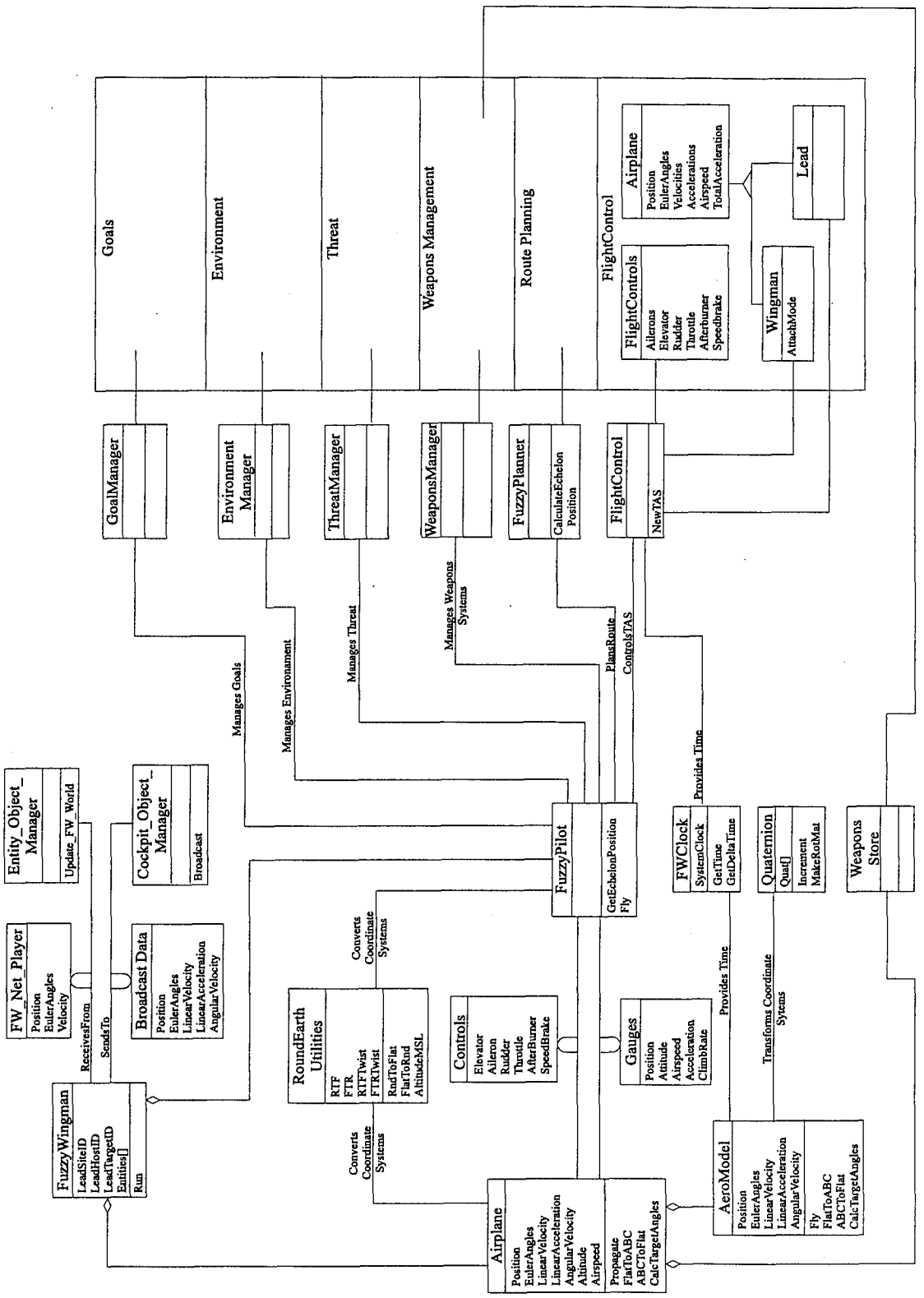

Figure 4-3 Automated Wingman Object Model

4.3.2 Dynamic Model

The next step in the design of the basic architecture is the State Transition Diagram shown in Figure 4-4. This model shows the states through which the basic architecture traverses. At the beginning of every loop, the Automated Wingman obtains an update of the lead aircraft's coordinates from the Entity_Object_Manager. It then feeds that information to the FuzzyPilot, which uses that information and data from the Airplane object to determine new throttle and stick (TAS) settings. Once the FuzzyPilot has completed its cycle, the new TAS settings are given to the Airplane object for use in the aerodynamics model. The Airplane object then "flies" the plane, updating the coordinates and dead-reckoning parameters. Finally, the Automated Wingman gives the new data to the Cockpit_Object_Manager, which determines if a new broadcast is

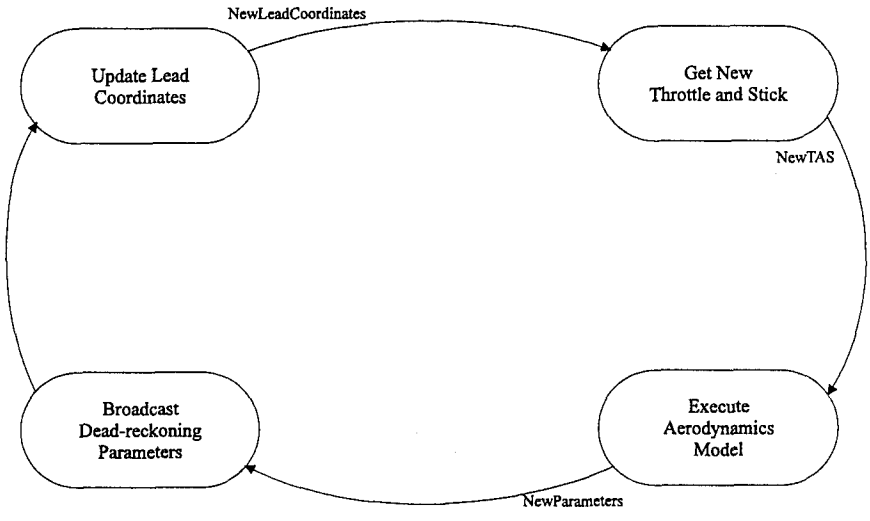

Figure 4-4 State Transition Diagram of the Automated Wingman

required and takes the appropriate action. This behavior repeats as long as the Automated Wingman is running, allowing the Automated Wingman to make decisions, fly and participate in distributed simulations.

4.3.3 Airplane Object Design

The Airplane object represents the aircraft that the Automated Wingman is designed to fly. As shown in Figure 4-5, the Airplane object is an aggregate of at least two other objects, the AeroModel and the WeaponsStore. As the project matures and requirements are further refined, the aircraft can be augmented with more objects. However, aerodynamics and weapons are the two basic necessities of any combat aircraft and were therefore chosen to fill out the initial design.

The purpose of the AeroModel object is to provide the Automated Wingman with the flight dynamics of a real airplane. The aerodynamics model used is the one designed by Cooke at the Naval Post-Graduate School [COOK92]. Cooke's model is also the

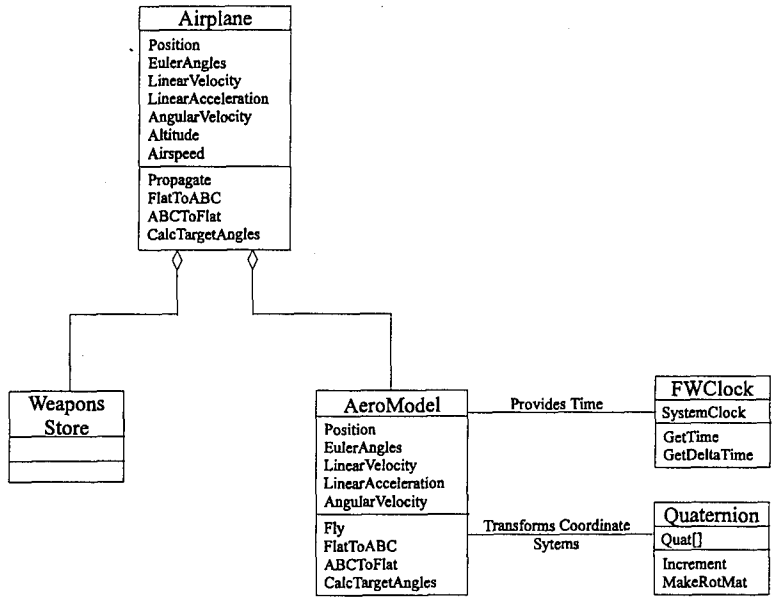

Figure 4-5 Airplane Object Diagram
(extracted from Figure 4-3 Automated Wingman Object Model)

model currently used in the AFIT Virtual Cockpit (VC) and was therefore a natural choice for this project. However, using the design from the VC was not possible because of its heavy reliance on the Performer graphics libraries that are specific to the SGI architecture. Hence, an extensive redesign of the container objects and a re-implementation of the code was required to achieve a clean, reusable interface. Since container objects are an implementation detail, further discussion of this topic will be deferred to the next chapter. However, the object model does show that the AeroModel object uses other objects to manage the system clock and a quaternion object. Unfortunately, different operating systems provide different function calls to access the system clock. The SystemClock object encapsulates the clock so that these differences remain transparent to the Automated Wingman. The Quaternion object, on the other hand, provides an efficient way of incrementally updating the euler angles that transform aircraft body coordinates into flat earth coordinates. This will be discussed in detail in the section 4.6. Once the redesign of the AeroModel object was complete, the new version of the AeroModel object met the requirement for a machine independent architecture while providing realistic flight dynamics.

The WeaponsStore object is designed to be a simple data record of the weapons carried by the Automated Wingman during a given mission. The FuzzyPilot would access this information to find out what weapons remain at his disposal. Its design is simple and does not warrant further discussion.

The State Diagram in Figure 4-6 shows the state transitions within the Airplane object. There are only two states, idle and flying. When an updated TAS setting is given to the Airplane, it passes that information to the AeroModel and waits for it to return with updated coordinates and dead-reckoning parameters. If the Airplane receives a request for weapons information, it services that request by accessing the WeaponsStore object and then continues to wait for more information. This continues for as long as the Automated Wingman is in operation.

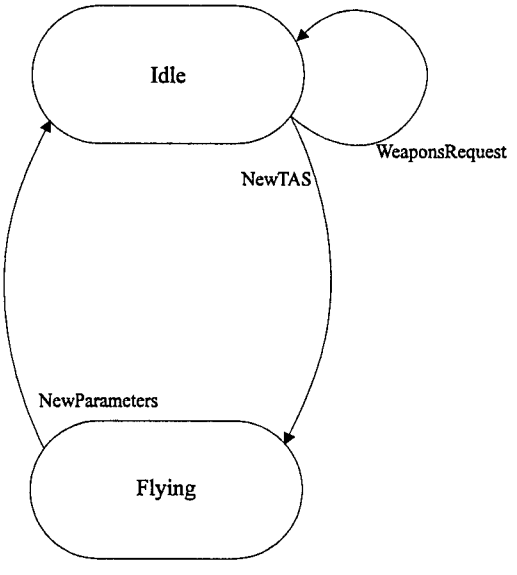

Figure 4-6 Airplane State Transition Diagram

4.3.4 FuzzyPilot Object Design

The second major object in the Automated Wingman is the FuzzyPilot object. This object provides the Automated Wingman with an interface to the fuzzy expert system. Most of the design for this object decomposes the functions of a pilot into categories that can be implemented using an expert system shell and pilot specific knowledge. These functions will be discussed in this section.

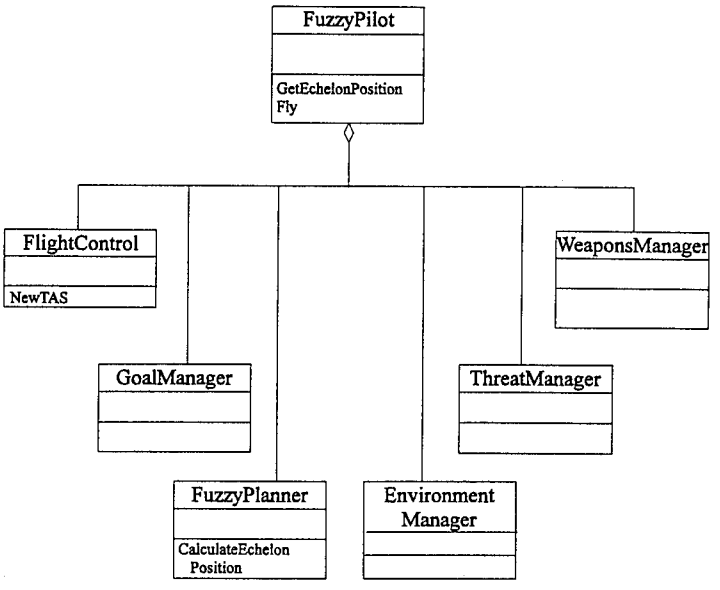

Figure 4-7 FuzzyPilot Object Model

The lowest level of functionality is in the FlightControl object. FlightControl provides the FuzzyPilot with the knowledge necessary to fly the airplane to a new location. This is called "fly-to-point" operation. FlightControl keeps track of location, orientation (euler angles), velocity vector, acceleration vector and other factors (see Table 4-1) of both the Automated Wingman and the lead aircraft. It also keeps track of and interfaces with the Controls object, often called the TAS (Throttle And Stick). To accomplish this, the FlightControl module requires information about the current status of the Automated Wingman, the current status of the lead aircraft simulator, the TAS, and coordinates of the new point in space to fly to. The outputs of the FlightControl object are new settings for the TAS components shown in Table 4-1. FlightControl is a low level function, but it is the foundation upon which all other functions of the FuzzyPilot are built.

Table 4-1 Parameters Maintained by the Flight Control Object

Wingman	Lead	TAS	Miscellaneous
Attach Mode	Position	Aileron	New Target Position
Position	Euler Angles	Elevator	Bearing to Target Position
Euler Angles	Velocity Vector	Rudder	Range to Target Position
Velocity Vector	Acceleration Vector	Throttle	
Acceleration Vector	Airspeed	Afterburner	
Airspeed		Speedbrake	

The next level is the FuzzyPlanner. The purpose of the FuzzyPlanner is to provide the FlightControl module with the coordinates of the new point to which it should fly. FuzzyPlanner does this based on the current orders under which the Automated Wingman is operating. For example, if flying in formation, the FuzzyPlanner uses the current coordinates of the lead aircraft to calculate the coordinates for the Automated Wingman. The FlightControl module then attempts to minimize the difference between the desired and actual coordinates. The FuzzyPlanner is also knowledgeable about rolls, "S" turns, and other maneuvers that occur over a period of time. This module decomposes these maneuvers into a series of points and keeps track of where the Automated Wingman is in the maneuver. As the maneuver progresses, the FuzzyPlanner feeds the appropriate points to the FlightControl module for action. This provides the Automated Wingman with a believable and extendible maneuver set with which to work.

The next three modules reside at the same functional level. They are the Weapons Manager, the Threat Manager, and the Environment Manager. Each of these manage the area indicated by their names. The Weapons Manager uses information about the current target such as type, range, bearing, etc., to select the most appropriate weapon from the

WeaponsStore. The Environment Manager keeps the most up to date information about weather, terrain, clouds, etc., that might affect pilot decisions. The Threat Manager watches the environment for unfriendly aircraft, Surface-to-Air Missile sites, and other potential dangers. All of these provide input to the next level for use in goal determination.

The top level of the FuzzyPilot is the Goal Manager. The Goal Manager's objective is to keep the Automated Wingman on track and performing its mission by interfacing with the Mission Knowledgebase (see Section 4.4.3). To do this, it integrates all the data obtained from its sub-levels and other information, such as voice commands, and chooses the appropriate goal to match the situation. It then selects a maneuver to satisfy the chosen goal and instructs the FuzzyPlanner module to fly that maneuver. It also tells the Weapons Manager when to fire its ordinance and the Threat Manager when to release chaff or flares to confuse an incoming threat. The Goal Manager is therefore a key component of the FuzzyPilot and will be discussed further in the Section 4.4.3.

4.4 Expert System Design

The fuzzy expert system provides the Automated Wingman with the intelligence and decision making capability required to perform its mission. It is the subject of careful design. First, a fuzzy logic production system was chosen for its ability to mimic human reasoning [KOSK93a]. Then, the overall problem was decomposed into a hierarchy of knowledgebases that cover the problem of flying a combat fighter. This led to the design of the blackboard system shown in Figure 4-9. Next, several of the knowledgebases were designed, including a mission goal tree. Finally, the lowest level knowledgebase,

FlightControl, was fully designed and prepared for implementation. The factors for arriving at these design decision are discussed in this section.

4.4.1 Knowledgebase Hierarchy

The knowledge required to fly and fight in a combat aircraft is tremendous. In order to make the problem manageable, I decomposed it into a hierarchy of knowledgebases. The hierarchy is shown in Figure 4-8. At the highest level is the mission knowledgebase. This knowledgebase guides the overall action of the Automated Wingman. Threat is part of the mission knowledgebase because when the wingman is threatened, it will change its mission to evade or suppress the threat. Tactics, Weapons, and Environment compose the next level of the hierarchy. Once the mission is selected, these knowledgebases analyze the environment and select the tactics and weapons required to fulfill the mission. The Maneuver knowledgebase then selects the appropriate aircraft maneuvers for the chosen tactics and weapons. Finally, the FlightControl

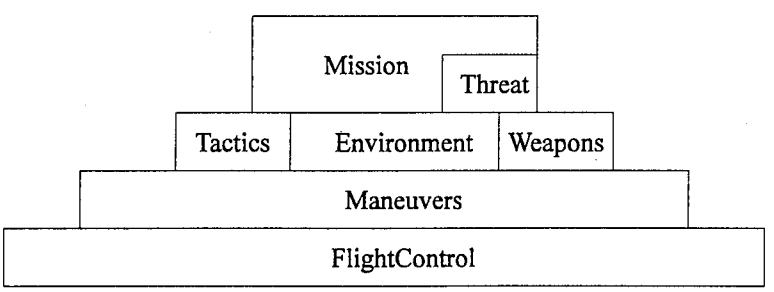

Figure 4-8 The Knowledgebase Hierarchy

knowledgebase implements the selected maneuvers. This hierarchy forms the basis for the operation of the Automated Wingman.

4.4.2 Blackboard System Design

Production systems function by matching rule antecedents, the "if" part, to known facts and then asserting the rule consequents, the "then" parts. The facts are stored in a data structure called the blackboard. Depending upon the design, either all, some, or none of the facts may be visible to all of the productions. The scope of the facts is up to the designer. For the Automated Wingman, I chose a two blackboard approach to maintain a separation between the two domains I identified as important, aircraft systems and the environment. The design is shown in Figure 4-9. The smaller of the two blackboards is the Environment blackboard. This data structure maintains information that is external to the Automated Wingman, such as other entities, threats, and weather. Information on this blackboard is reduced and the result introduced, through the appropriate modules, to the main blackboard. All reasoning about flying the airplane takes place on the main blackboard, called the Aircraft blackboard. Each of the objects discussed so far has an interface with the blackboards and uses the information that it keeps. However, each object also has a private section of the blackboard, not shown, for maintaining its own working facts. The blackboards ensure that each component has access to the information it needs without bogging down the system with facts that are not required by all.

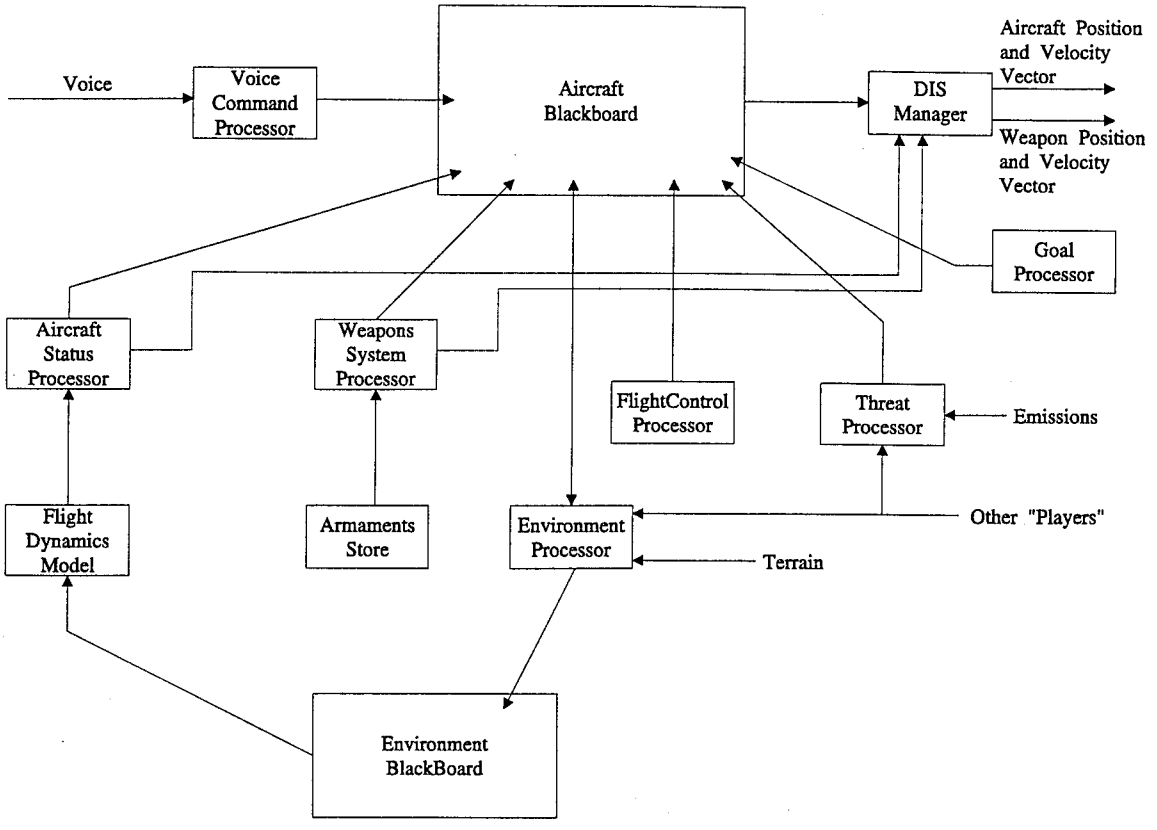

Figure 4-9 System Blackboard Design

4.4.3 The Mission Knowledgebase

The Automated Wingman is guided by the goal hierarchy in Figure 4-10, which is the central part of the Mission knowledgebase. This hierarchy shows the activity of the Automated Wingman at a high level and is therefore incomplete. Completing this graph involves extensive studies of pilot behavior and is left for future research. However, even in its current state, the graph demonstrate the types of goals that the Automated Wingman can select from. Below each of the lowest nodes shown there are more goals that lead to certain tactics and maneuvers. The Automated Wingman navigates its way, based on the current situation, down to the maneuver level and instructs the FuzzyPlanner to carry out the chosen maneuver. Then, on every loop described by Figure 4-4, the Automated

Wingman compares the environment and the current state and determines whether the same goal still applies or if it is time to select a different activity. This way, the Automated Wingman is not committed to a particular tactic and can break off should the circumstances warrant it. This provides the Automated Wingman with a powerful guiding intelligence that may fool other, manned, entities into believing that it is under human control.

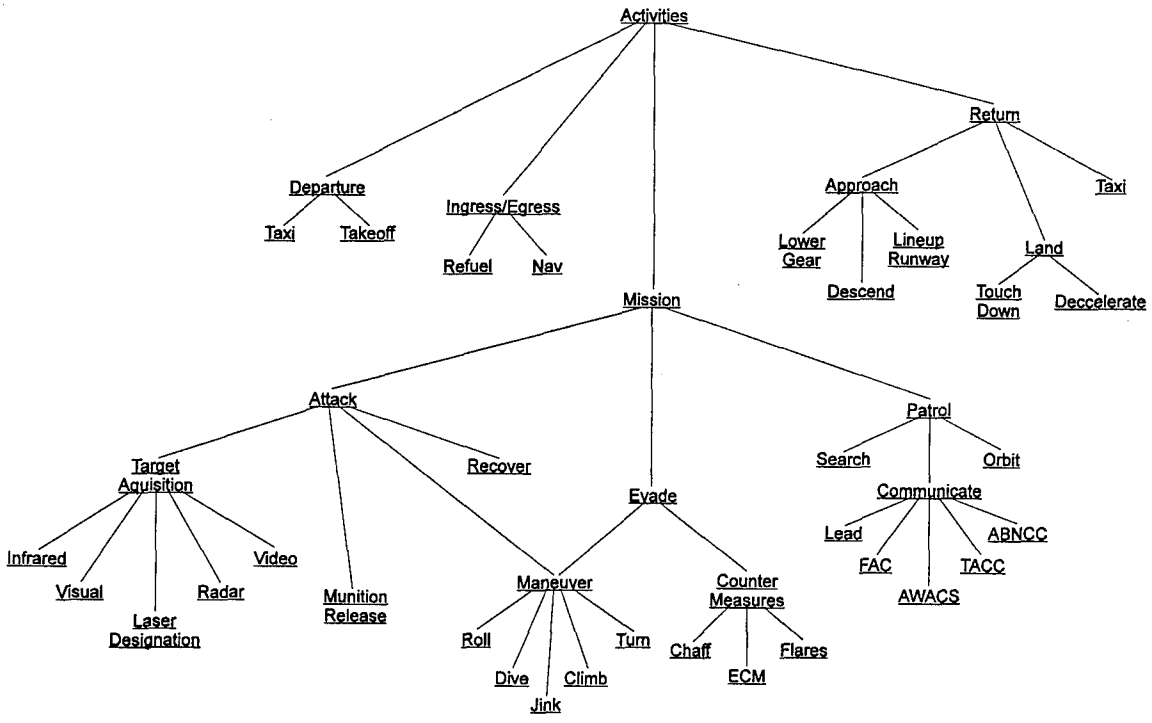

Figure 4-10 High-Level Goal Hierarchy

Another point in favor of this design choice is that as the project matures it is easy to add more knowledge to the system. Simply changing and recoding the goal tree is all that is required. Assuming that the aerodynamics model can handle the maneuvers, this makes developing and testing new tactics much easier to accomplish then hard-coding the knowledge into the system. Therefore, the use of a goal tree such as Figure 4-10 adds for a powerful reasoning capability to the Automated Wingman.

4.4.4 The Environment Knowledgebase

The Environment knowledgebase manages the Automated Wingman's knowledge of its surroundings through the Environment linguistic variable. This linguistic variable is the domain of the Environment Manager of the FuzzyPilot. The design is shown in Table 4-2 and Figure 4-11. In designing this linguistic variable, I have used a modified version of the Object Modeling Technique found in [RUMB91]. The boxes represent linguistic objects, i.e., variables, and the attributes are the term sets describing that object. Links between linguistic objects show how the linked objects relate. The triangle shows

Table 4-2 Environment Linguistic Variables

Linguistic Variable	Description
Turbulence	Air turbulence in the vicinity.
Snow	Severity of snow falling in the vicinity of the Wingman
Rain	Severity of rain in the immediate vicinity of the Wingman
Hail	Severity of hail in the immediate vicinity of the Wingman
Clouds	Cloud cover in the immediate vicinity of the Wingman
Fog	Severity of fog in the immediate vicinity of the Wingman
Haze	Severity of haze in the immediate vicinity of the Wingman
Thunderstorms	Severity of thunderstorms in the vicinity of the Wingman
Lightening	Severity of lightening in the immediate vicinity of the Wingman
Sun Position	The position of the sun relative to the Wingman
Moon Position	The position of the moon relative to the Wingman
Wind Direction	The direction of winds in the immediate vicinity of the Wingman
Wind Intensity	The speed of both steady wind and gusts around the Wingman
Illumination	The level of light surrounding the Wingman
Terrain	The type of terrain over which the Wingman is flying
Visibility	How well the Wingman can see through the environment

class hierarchy and inheritance. The environment itself consists of several of these linguistic objects instead of being a single linguistic variable. This design shows how the objects relate and affect each other. For example, the snow object inherits a DIS Value

(as yet undefined) and derives from it a fuzzy value. That fuzzy value is described by one or more of the following term sets; flurries, light, medium, or heavy. Light snow lowers visibility from, perhaps, unlimited to limited, depending on the other factors involved with the determination of visibility. Visibility, a fuzzy linguistic variable with three term sets, is stored as part of the environment for access by the Automated Wingman. Positions, such as the sun and moon, are fuzzified to allow them to be dealt with in general terms. Unfortunately, DIS does not as yet contain the mechanisms to support this capability. As both DIS and the Automated Wingman mature, however, more work can be done in this area to enhance the training effectiveness of DIS and the Automated Wingman.

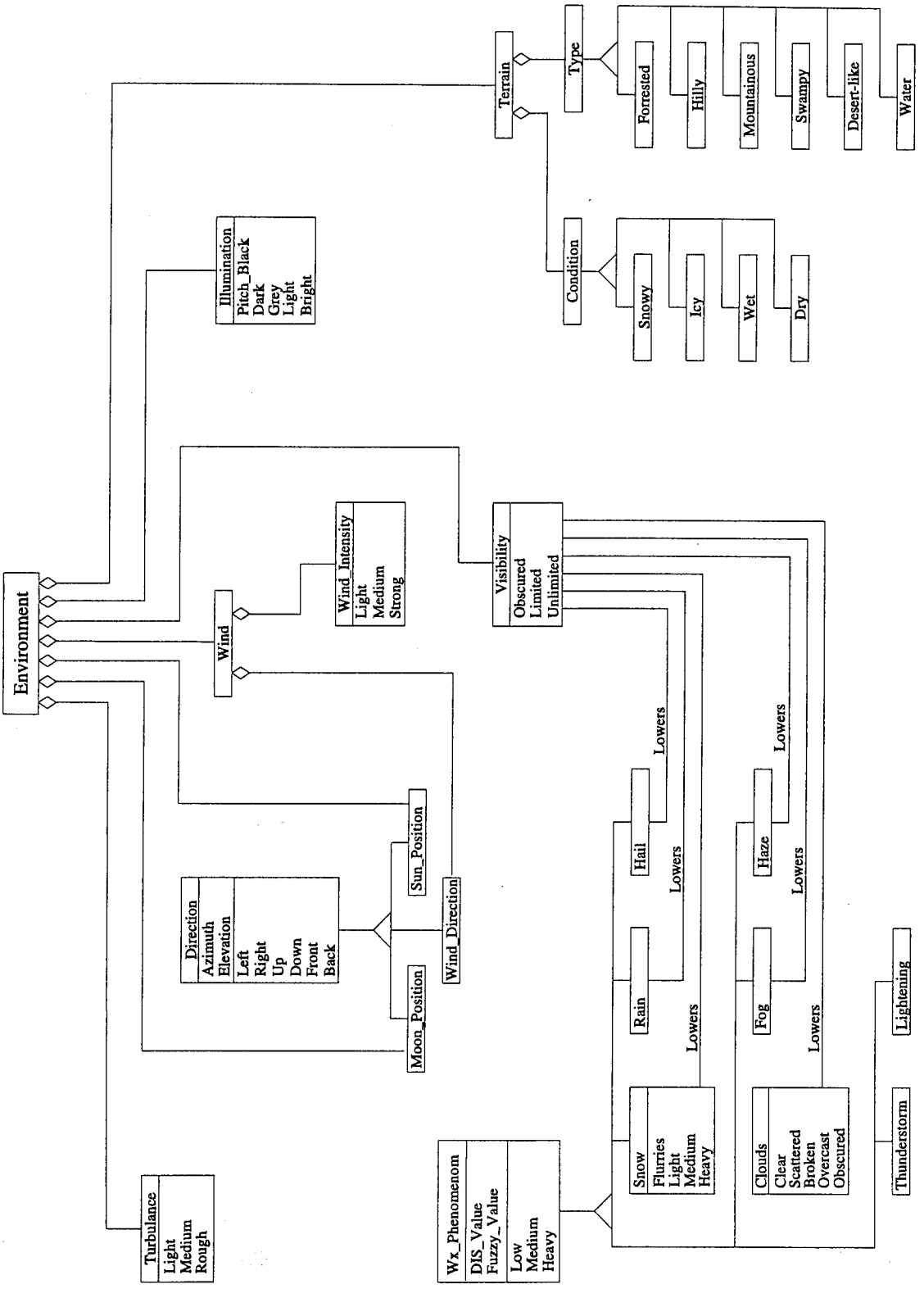

Figure 4-11 The Environment Linguistic Variable

4.4.5 The Threat Knowledgebase

Like the Environment linguistic variable, the design of the Threat linguistic variable also uses the modified OMT style. Its design, Table 4-3 and Figure 4-12, also shows how objects in the simulated world affect the threat level of the Automated Wingman. For example, an SA-6 Surface-to-Air Missile increases the RadarThreat, which may be decreased by launching chaff and employing Electronic countermeasures (ECM). The RadarThreat linguistic variable then affects the overall Threat level which also uses other factors to arrive at an overall threat level determination. This capability is also more advanced than the Automated Wingman requires at this time. However, the primary job of a wingman is to keep the lead informed of threats and other environmental

Table 4-3 Linguistic Variables Used in Threat Assessment

Linguistic Variable	Description
Threat	The super-class of threat types.
Surface-to-Air	Surface-to-Air Threats
Air-to-Air	Air-to-Air Threats
SA-2 thru SA-14/16	Surface-to-Air missile threats (some IR guided, some radar guided)
AA-2 thru AA-9	Air-to-Air missile threats
Combat Aircraft	Combat aircraft in the vicinity of the Wingman
Friendly	Friendly combat aircraft in the vicinity of the Wingman
Enemy	Enemy combat aircraft in the vicinity of the Wingman
IR Threat	Threat due to Infra-Red seekers
Radar Threat	Threat due to radar guided missiles
Cannon	Unguided threat from enemy combat aircraft
Chaff	A countermeasure against radar guided threats
Flares	A countermeasure against IR guided threats
ECM	A countermeasure against radar guided threats
Visibility	Affects IR seekers

concerns and at some future point more work will be required in this area

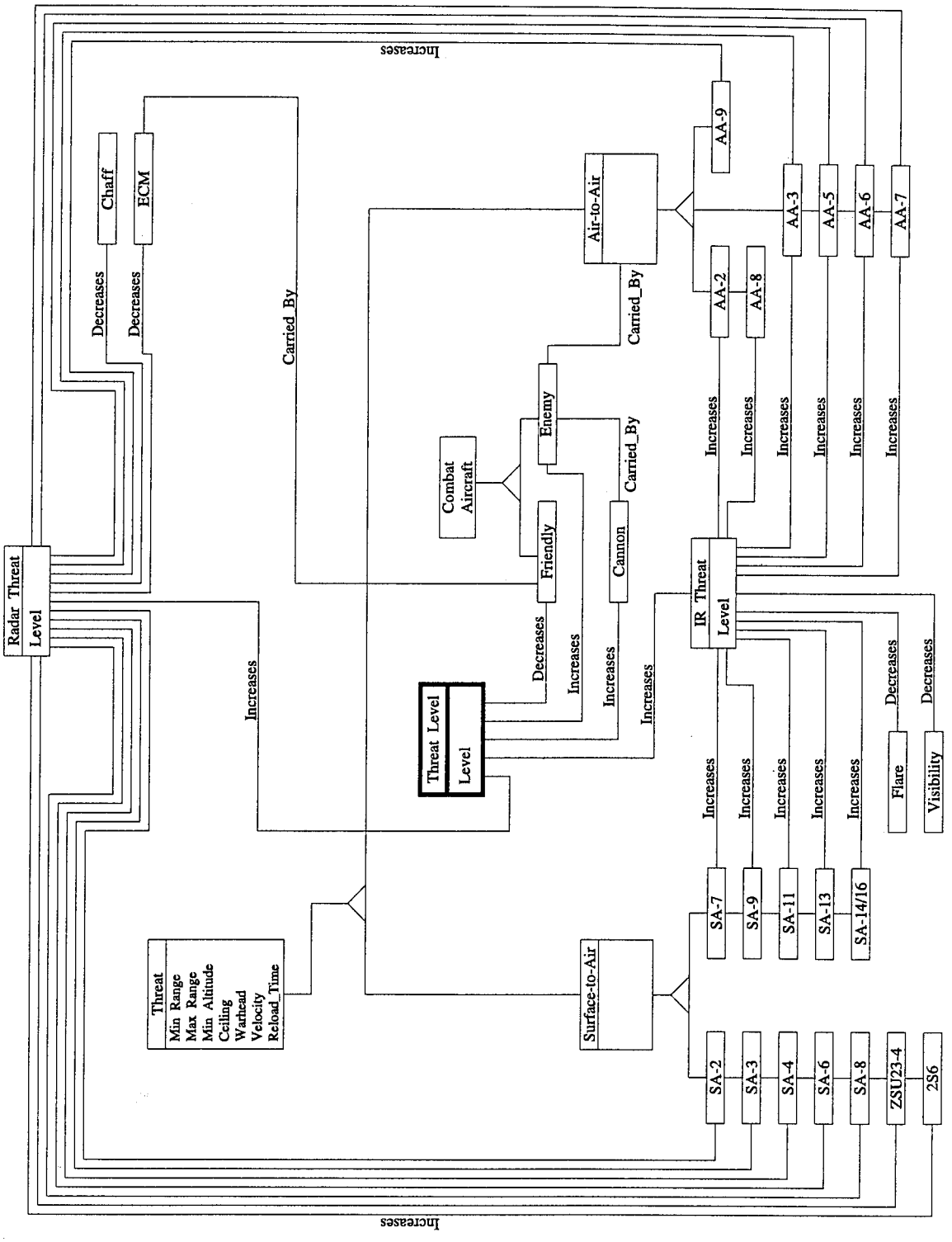

Figure 4-12 The Threat Linguistic Variable

4.4.6 Flight Control Knowledgebase

While fuzzy knowledgebases are used throughout the entire knowledgebase hierarchy, the initial version of the Automated Wingman focuses mostly on the lower level knowledgebase called FlightControl. The FlightControl knowledgebase consists of a set of linguistic variables and a set of rules that relate these linguistic variables to the state of the Automated Wingman and the action that the Automated Wingman needs to take. This section describes the design process that I followed and presents and discusses my final design.

4.4.6.1 Linguistic Variable Design

The design of the FlightControl knowledgebase was an evolutionary process. My first attempt concentrated on developing two flight modes, called "fly-to-point" and "fly-to-angles". The idea behind this concept was that when the wingman is far away from the lead, the wingman's primary goal is to get close to the lead using any method possible. The best way to do that is to calculate the control inputs required to fly to the point in space where the lead aircraft is at the current time, hence the name "fly-to-point". The lead's position is constantly updated so as the wingman closes in on it, the wingman updates it's trajectory to match the lead. As the trajectories of the two entities become the same, the second flight mode, fly-to-angle, takes over. In this mode, the wingman attempts to match the euler angles of the lead and maintain the matching trajectory. Therefore, if the lead begins to climb or bank, the wingman will follow. Should the wingman lose the formation position, the "fly-to-point" rules take over as the range between the two increases and, once again, the wingman is brought back to the lead.

Using these two modes seemed like a good methodology to keep the two entities flying in formation.

Unfortunately, the "fly-to-point"/"fly-to-angles" method had a fatal flaw. The fly to point rules were always chasing the tail of the lead aircraft and the trajectories never came together. This caused the trajectory of the Automated Wingman to "porpoise" out of control with ever increasing positional errors. At that time, I realized that the wingman had to calculate not how to get to where the lead is now, but how to get to where the lead will be at some future time. I projected variables dealing with velocity and acceleration into the future using a value called a "lookahead". Initially, I used a fixed period of time, but I found that while projecting 15 seconds was good for maintaining formation, it was too long for attempting to achieve the formation position. I also found that 5 seconds was good for achieving the formation position but resulted in over-control and rapid oscillations while flying in formation. Therefore, I decided to vary the lookahead period based on the range. When range is greater than 100 meters, the Automated Wingman uses a 5 second lookahead, when it is less than 50 meters away, it uses a 15 second lookahead. In between 50 and 100 meters, the lookahead period is a linear function of distance between 5 and 15 seconds. The results (see Figure 6-1) show that this was a good scheme.

Once I projected variables into the future, I found that there was no longer a need for a "fly-to-angles" mode. This is because projecting the velocity values into the future takes into account any difference in euler angles. So, I abandoned the "fly-to-angles" mode and developed the "fly-to-point mode only.

The final design of the FlightControl knowledgebase implements the 15 linguistic variables shown in Table 4-1. The names of each of the linguistic variables were chosen to describe the purpose of that variable. Variables with "Current" in the name describe quantities as they are now while "Projected" variables use the variable lookahead period.

Table 4-1 Linguistic Variables in the Automated Wingman

Linguistic Variables
Current Relative Altitude
Projected Relative Altitude
Vertical Velocity
Vertical Velocity Difference
Desired Vertical Velocity Difference
Projected Vertical Velocity Difference
Vertical Acceleration
Total Acceleration
Current Relative Airspeed
Projected Relative Airspeed
Projected Airspeed Difference
Relative Heading
Range
Lead Bearing
Bank Angle

A common concern with all of the term sets of all of the linguistic variables is determining the membership functions to be used. Although Figure 2-1 shows non-linear membership functions for the term sets of Age, these functions can be computationally expensive. Therefore, I have chosen to stay with linear functions. The exact definition of each of the membership functions is ad-hoc. So, I have used my best guess in consideration of the purpose of the linguistic variable being designed. For example, I chose 0 to 100 meters to represent "Nil" for the linguistic variable Range. This seemed appropriate given the distance errors that can naturally occur when using DIS.

Relative Altitude

There are two linguistic variables used to describe relative altitude. These are CurrentRelativeAltitude and ProjectedRelativeAltitude. CurrentRelativeAltitude describes the altitude difference between the Automated Wingman's current position and where the Automated Wingman should be. ProjectedRelativeAltitude describes the relative altitude between the two at some time, depending on the range, in the future given that the current velocities and accelerations remain constant. The design of Relative Altitude is shown in Table 4-2 and Figure 4-13.

Table 2 Relative AltitudeRelative Altitude

Altitude Difference (meters)	Lower	Nil	Higher
-75	1	0	0
-50	1	0	0
-30	0	0.4	0
0	0	1	0
30	0	0.4	0
50	0	0	1
75	0	0	1

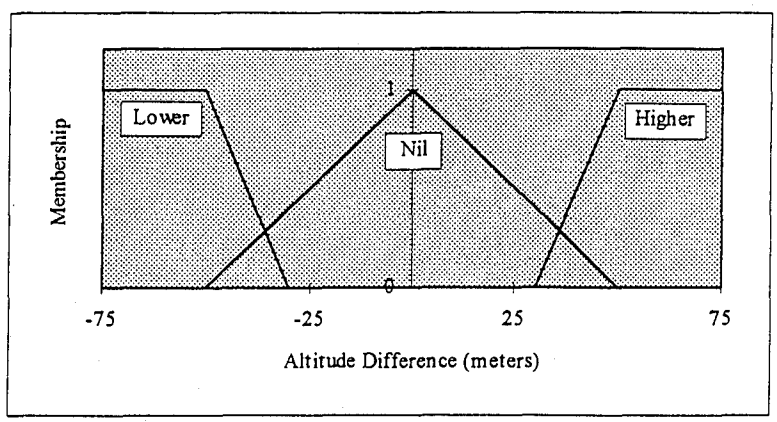

Figure 4-13 Relative Altitude Graph

4.4.6.1.2 Vertical Velocity

The design of the linguistic variable VerticalVelocity is shown in Table 4-3 and Figure 4-14. VerticalVelocity describes the rate at which the Automated Wingman is climbing or diving and is used to determine the change in vertical velocity required to achieve the desired altitude.

Table 4-3 Vertical Velocity

Vertical Velocity			
Climb Rate (M/S)	Diving	Nil	Climbing
-5	1	0	0
-2	1	0	0
-1	0.5	0.5	0
0	0	1	0
1	0	0.5	0.5
2	0	0	1
5	0	0	1

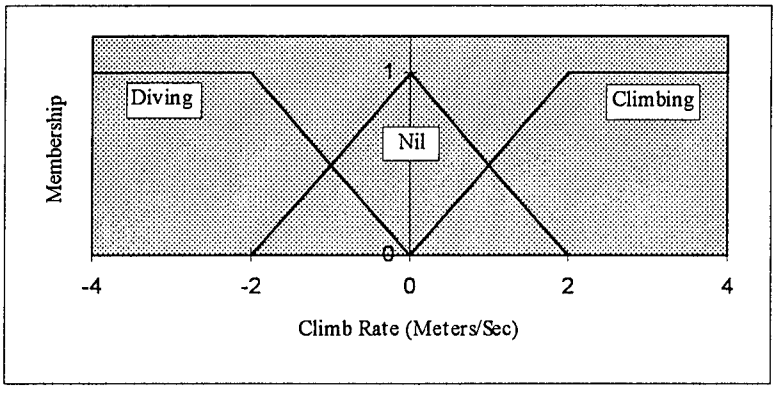

Figure 4-14 Vertical Velocity

4.4.6.1.3 Vertical Velocity Difference

There are three linguistic variables dealing with differences in vertical velocity. VerticalVelocityDifference describes the current difference in vertical velocities between the Automated Wingman and the lead simulator or the desired position from the FuzzyPlanner (see section 4.3.4). ProjectedVerticalVerticalDifference describes that difference projected 5 to 15 seconds, based on range, into the future. DesiredVerticalVelocityDifference indicates how the Automated Wingman should change its vertical velocity in order to minimize the projected difference. The design of these linguistic variables is shown in Table 4-4 and Figure 4-15.

Table 4-4 Vertical Velocity Difference

Vertical Velocity Difference			
Delta Climb Rate (M/S)	Slower	Nil	Faster
-10	1	0	0
-5	1	0	0
0	0	1	0
5	0	0	1
10	0	0	1

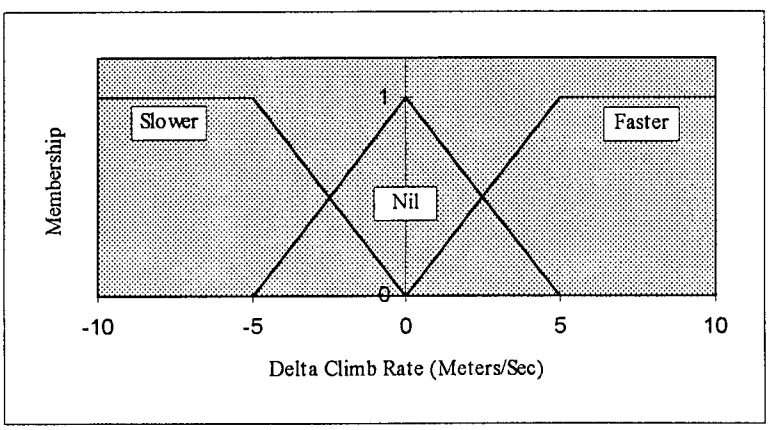

Figure 4-15 Vertical Velocity Difference Graph

4.4.6.1.4 Acceleration

Acceleration encompasses two linguistic variable, VerticalAcceleration and TotalAcceleration. VerticalAcceleration describes the change in climb rate of the Wingman while TotalAcceleration describes the change in airspeed. The design is shown in Table 4-5 and Figure 4-16.

Table 4-5 Acceleration

Vertical Acceleration			
Acceleration (M/S^2)	Negative	Nil	Positive
-5	1	0	0
-2	1	0	0
-0.5	0.25	0	0
0	0	1	0
0.5	0	0	0.25
2	0	0	1
5	0	0	1

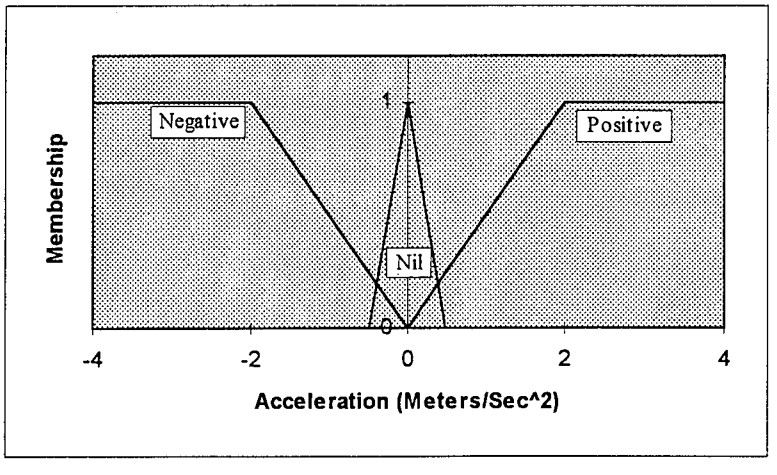

Figure 4-16 Acceleration Graph

4.4.6.1.5 Relative Airspeed

There are three airspeed linguistic variables. These are CurrentRelativeAirspeed, ProjectedRelativeAirspeed, and ProjectedAirspeedDifference. The first two are the current and projected airspeed differences between the Automated Wingman and the Lead simulator. The projected airspeed difference is based on the velocities and acceleration remaining at their current values for 5 to 15 seconds, depending on range. Projected airspeed difference is the change in the Automated Wingman's airspeed relative to itself over the next 5 to 15 seconds. The design is shown in Table 4-6 and Figure 4-17.

Table 4-6 Relative Airspeed

Current Relative Airspeed			
Airspeed Difference (M/S)	Slower	Nil	Faster
-15	1	0	0
-10	1	0	0
0	0	1	0
10	0	0	1
15	0	0	1

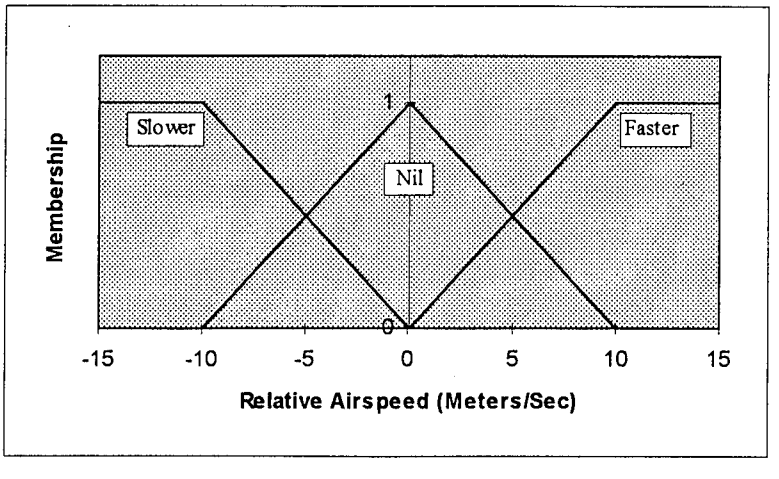

Figure 4-17 Relative Airspeed Graph

4.4.6.1.6 Relative Heading

The linguistic variable RelativeHeading describes the heading of the Lead simulator relative to the heading of the Automated Wingman. In other words, it indicates whether or not the two are traveling in the same direction. The design of RelativeHeading is shown in Table 4-7 and Figure 4-18.

Table 4-7 Relative Heading

Relative Heading				
Relative Heading (Deg)	Nil	Left	Right	Opposite
-180	0	0	0	1
-179	0	0	0	1
-90	0	1	0	0
0	1	0	0	0
90	0	0	1	0
179	0	0	0	1
180	0	0	0	1

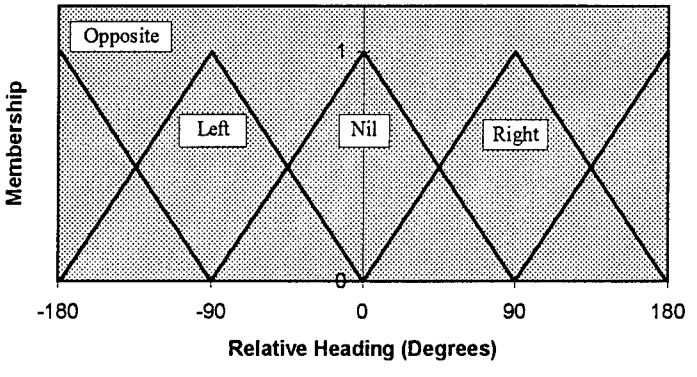

Figure 4-18 Relative Heading Graph

4.4.6.1.7 Range

The distance between the Automated Wingman and the Lead simulator, or new position from the FuzzyPlanner, is described by the linguistic variable Range. The design of Range is shown in Table 4-8 and Figure 4-19.

Table 4-8 Range

Range			
Range (meters)	Nil	Close	Far
0	1	0	0
1	1	0	0
50	0.505051	0	0
100	0	0.5	0
150	0	1	0
450	0	1	0
550	0	0	1
800	0	0	1

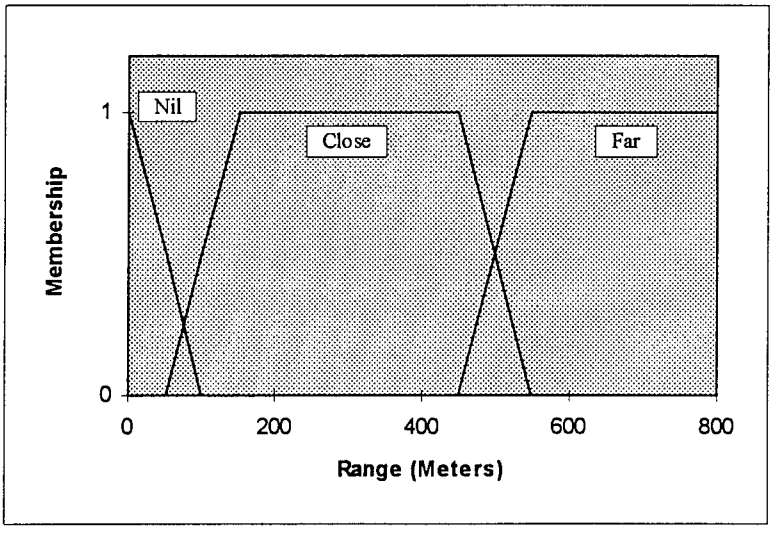

Figure 4-19 Range Graph

4.4.6.1.8 Lead Bearing

The direction in degrees of the lead simulator off the nose of the Automated Wingman is known as the lead bearing. In the Automated Wingman, LeadBearing is the linguistic variable that indicates where the lead simulator is in relation to the front of the Wingman For example, if the lead simulator is 100 meters in front of and 100 meters to the right of the Automated Wingman, then the lead bearing is 45 degrees The design of LeadBearing is shown in Table 4-9 and Figure 4-20.

Table 4-9 LeadBearing

Lead Bearing				
Bearing (Deg)	Front	Left	Right	Rear
-180	0	0	0	1
-179	0	0	0	1
-90	0	1	0	0
0	1	0	0	0
90	0	0	1	0
179	0	0	0	1
180	0	0	0	1

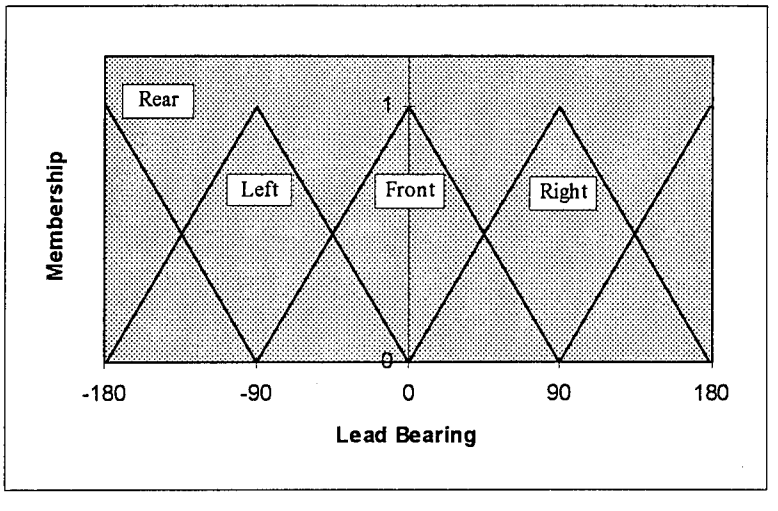

Figure 4-20 LeadBearing Graph

4.4.6.1.9 Bank Angle

The final linguistic variable is BankAngle. BankAngle describes the amount and direction of the current bank. The Automated Wingman can bank left or right or fly level. These are reflected in the design of BankAngle shown in Table 4-10 and Figure 4-21.

Table 4-10 BankAngle

Bank Angle			
Bank	Left	None	Right
-180	0	0	0
-179	0	0	0
-90	1	0	0
-89	0.988764	0	0
-1	0	0.988764	0
0	0	1	0
1	0	0.988764	0
89	0	0	0.988764
90	0	0	1
179	0	0	0
180	0	0	0

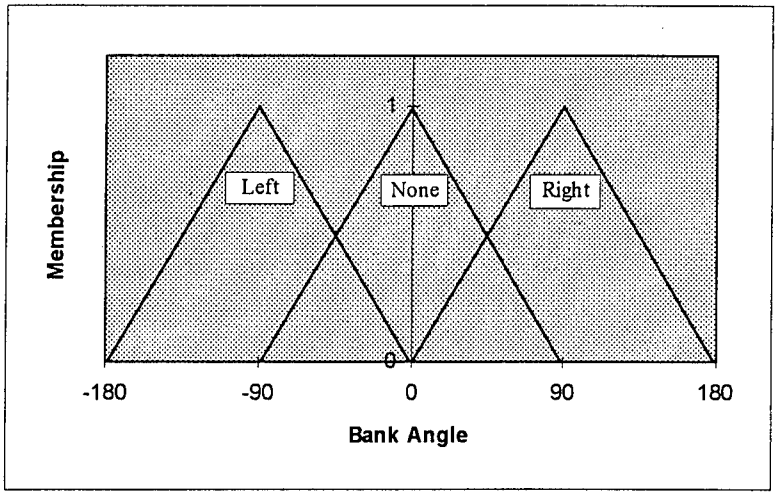

Figure 4-21 BankAngle Graph

4.4.6.2 Production Rule Design

There as three axis of control for the FlightControl knowledgebase. These are *altitude*, *heading*, and *thrust.* Using the linguistic variables designed in 4.4.6.1, I have designed production rule graphs for each of these axes of control. These rule graphs show how the linguistic variables combine to describe the state of the Automated Wingman along each axis and the correct action that the wingman should take. Each design is presented in this section

4.4.6.2.1 Altitude

Altitude is controlled in the FlightControl module of the FuzzyPilot and is described in terms of two linguistic variables, ClimbRate and ElevatorDeflection. The first describes the climb rate of the aircraft required to achieve the desired altitude. It considers the current altitude relative to the desired altitude and what the altitude difference will be at some time in the future given the current velocities and accelerations. ElevatorDeflection describes the change in elevator setting required to achieve the climb rate described above. Each of these relies on several linguistic variable in determining the appropriate value.

The Climb Rate Rule Graph is shown in Figure 4-22. Each of the bubbles represents a linguistic variable describing a quantity relevant to climb rate. The FlightControls module determines values for all these linguistic variables and then navigates this graph to determine the value for ClimbRate. For example, at the top of the graph the current value of AttachMode (see Figure 5-2) is checked. If AttachMode is

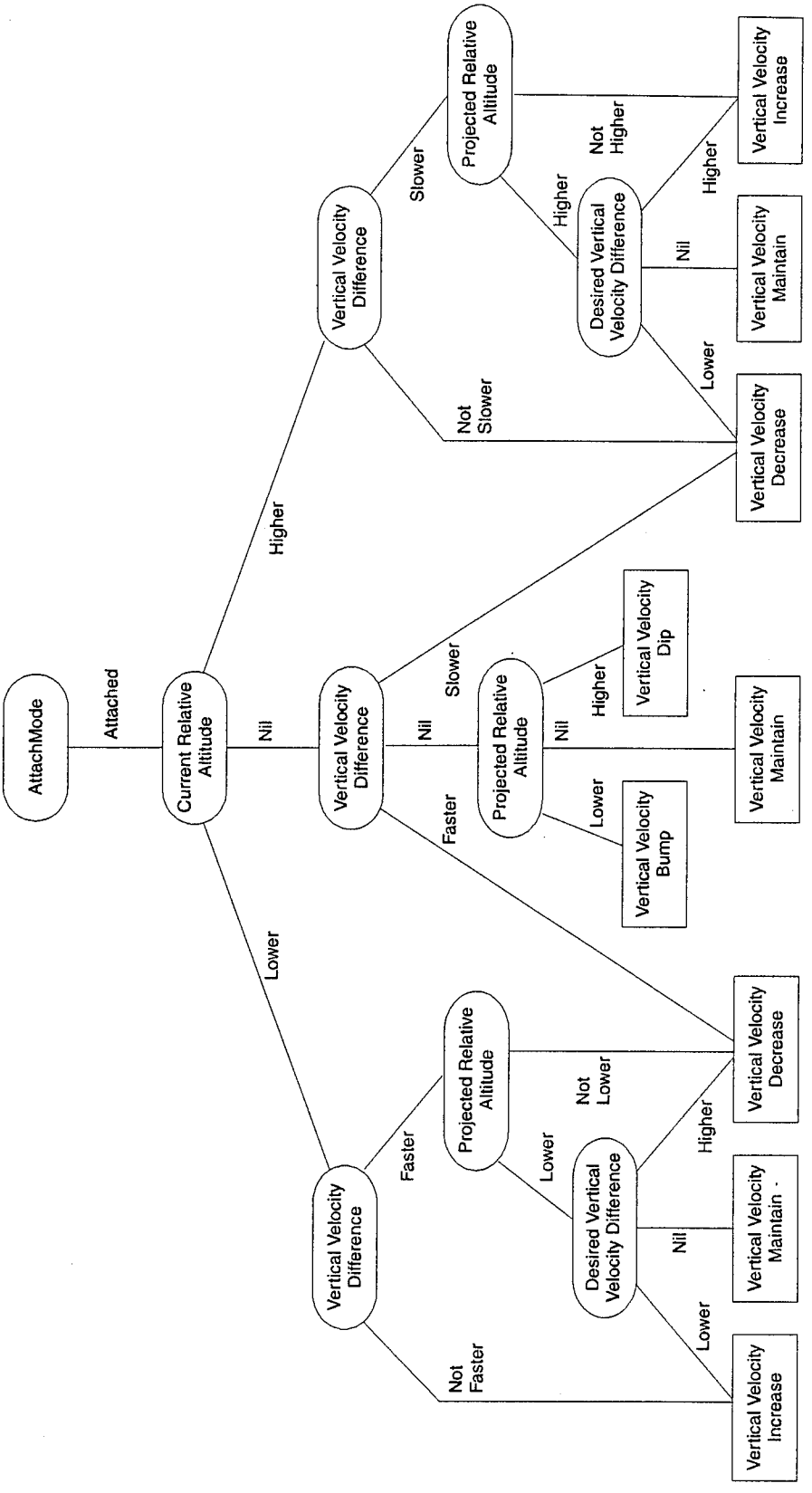

Figure 4-22 Climb Rate Rule Graph

"Attached" then check the linguistic variable "CurrentRelativeAltitude". If that is "Nil" then check the VerticalVelocityDifference. If that is "Nil" also, then check the ProjectedRelativeAltitude. A "Nil" there means maintain the current vertical velocity (climb rate). That is the "crisp" path. All these linguistic variables are fuzzy, and therefore, there may be other paths that apply but have less weight. This will "spread" the value of ClimbRate out so that it may encompass all five term sets, Decrease, Dip, Maintain, Bump, and Increase. In fact, in general, all five term sets will apply to some degree but most will apply to zero degree. The flexibility of the degree to which a term set describes the linguistic variable provides the power of this mechanism of reasoning.

The ElevatorDeflection linguistic variable operates in a manner similar to the ClimbRate linguistic variable. Its rule graph is shown in Figure 4-23. It uses the VerticalVelocity linguistic variable, a synonym for ClimbRate, to determine what action should be taken with the elevator control. At the top of the graph is the VerticalVelocity linguistic variable and its term sets, which lead down differing paths. Once again, all paths that apply to a degree greater than zero are activated. This leads to five term sets for the linguistic variable Elevator Deflection, Down, NudgeDown, Nil, NudgeUp, and Up. Once all the paths have been activated, the value of ElevatorDeflection is defuzzified to produce a single value that represents the amount of change required for the stick. That single value incorporates all the term sets that applied in every linguistic variable considered. Therefore, no information was ignored and a better decision results.

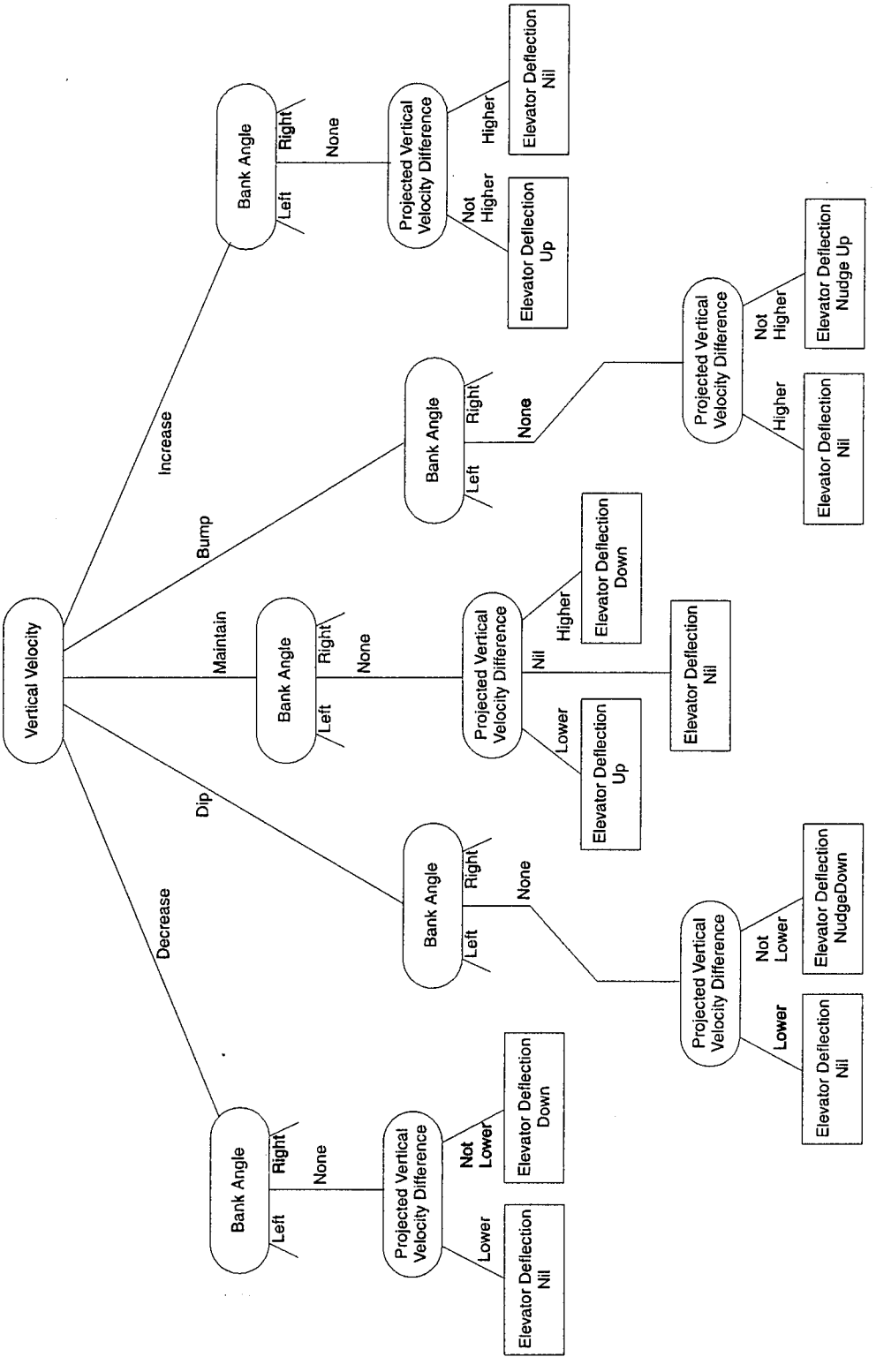

Figure 4-23 Elevator Deflection Rule Graph

4.4.6.2.2 Heading

Like altitude, heading is also controlled by linguistic variables. It is very similar to the process shown above and, hence, will not be discussed further.

4.4.6.2.3 Airspeed

Airspeed is another quantity of the FlightControl module that is determined by using linguistic variables. However, airspeed is different because the throttle is not the only input into its determination. It also relies on the afterburner and speedbrake. For this reason, there are three rule graphs for airspeed, as shown in Figure 4-24, Figure 4-25, and Figure 4-26. The airspeed rule graph generates the airspeed goal, which is used by the throttle and afterburner/speedbrake rule graphs. This layering of linguistic variables shows how one goal can be used to control several output variables

Figure 4-24 shows the rule graph used to develop the airspeed goal. Six separate linguistic variables are factored into the airspeed goal. Theses are described in Table 4-11. After reasoning with the fuzzy value from each of these linguistic variables, a fuzzy goal is calculated. Five term sets have been designed into the Airspeed Goal linguistic variable, each representing a possible goal. These are Decrease, Dip, Maintain, Bump, and Increase. As with the Vertical Velocity Goal, the airspeed goal is a fuzzy value and several of these term sets may apply to some degree. This could be defuzzified into one goal. However, that would discard information about the overall goal and be a less desirable response. Therefore, this fuzziness will be carried over into the decisions to be made about the throttle, afterburner, and speedbrake.

Table 4-11 Linguistic Variables Used to Determine the Airspeed Goal

Linguistic Variable	Description
Attach Mode	Describes whether or not the Automated Wingman is flying in formation
Relative Heading	The heading of the Lead with respect to the heading of the Automated Wingman
Range	The distance to the Lead from the Wingman
Current Relative Airspeed	The Leads airspeed with respect to Automated Wingman, described as a scalar quantity
Lead Bearing	Direction of the Lead off the Wingman's nose.
Projected Relative Airspeed	The Lead's projected airspeed relative to the Automated Wingman's projected airspeed at 5 to 15 seconds in the future.

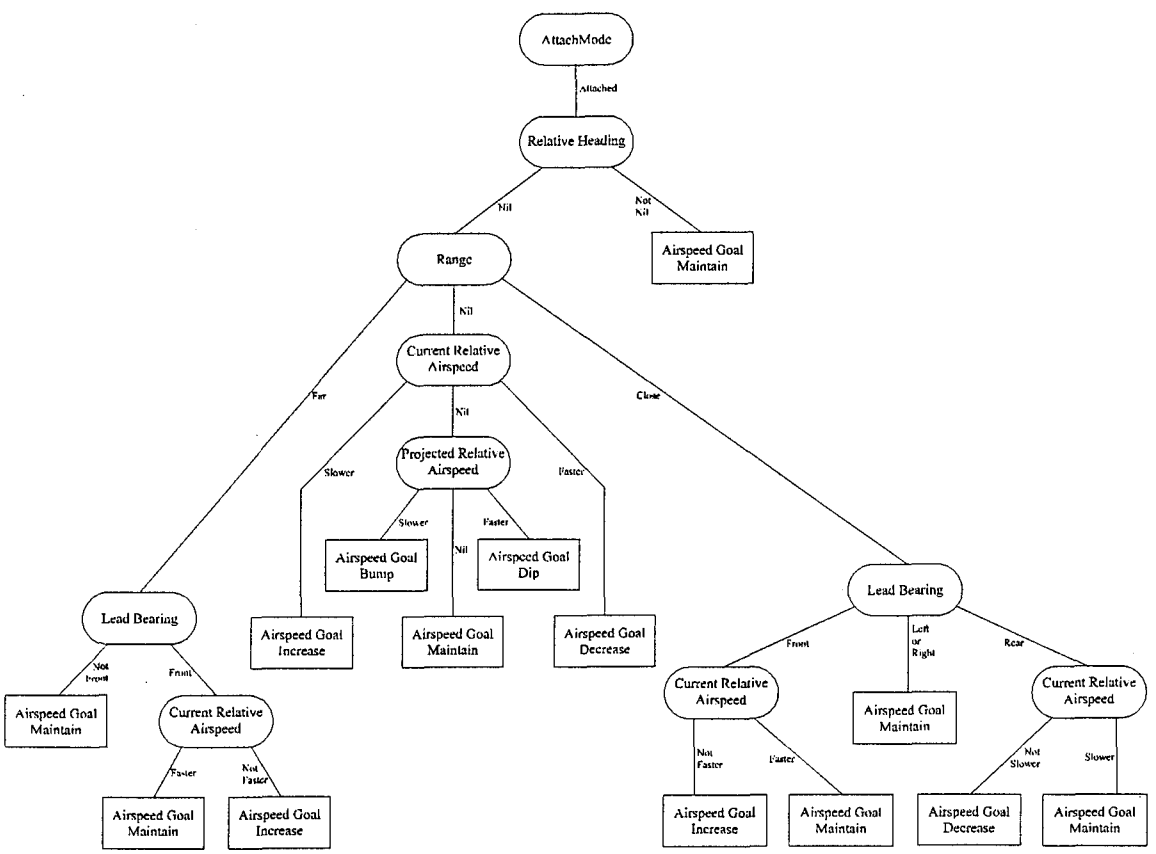

Figure 4-24 Airspeed Rule Graph

Once the Airspeed Goal has been set, it is used to determine the appropriate change to the throttle ,as shown in Figure 4-25, and to the afterburner and speedbrake, Figure 4-26. The throttle rule graph shows the linguistics variables considered in the decision about how to change the throttle setting. Again, several linguistic variables are factored into the decision for each of the throttle, the afterburner and the speedbrake. The linguistic variables used to determine the new throttle setting are described in Table 4-12 and those for the afterburner and speedbrake in Table 4-13. The speedbrake and afterburner graphs are a little different from the others. Instead of representing fuzzy quantities, the afterburner and speedbrake are either on or off, i.e., they are crisp. Therefore, there is no defuzzification. This shows the necessity of choosing a fuzzy expert system shell that is capable of non-fuzzy reasoning.

Table 4-12 Linguistic Variables used to Determine the Change in Throttle

Linguistic Variable	Description
Airspeed Goal	What action to take with the airspeed
Vertical Velocity	How fast is the airplane climbing or diving
Projected Airspeed Difference	Given the current velocities and accelerations, what will be the change in airspeed in the next 5 to 15 seconds.

Table 4-13 Linguistic Variables Used to Determine the Afterburner and Speedbrake Settings

Linguistic Variable	Description
Projected Airspeed Difference	Given the current velocities and accelerations, what will be the change in airspeed in the next 5 to 15 seconds.
Range	The distance to the Lead from the Wingman
Throttle Setting	The current throttle setting (must be > 0.95 for afterburner to be on)

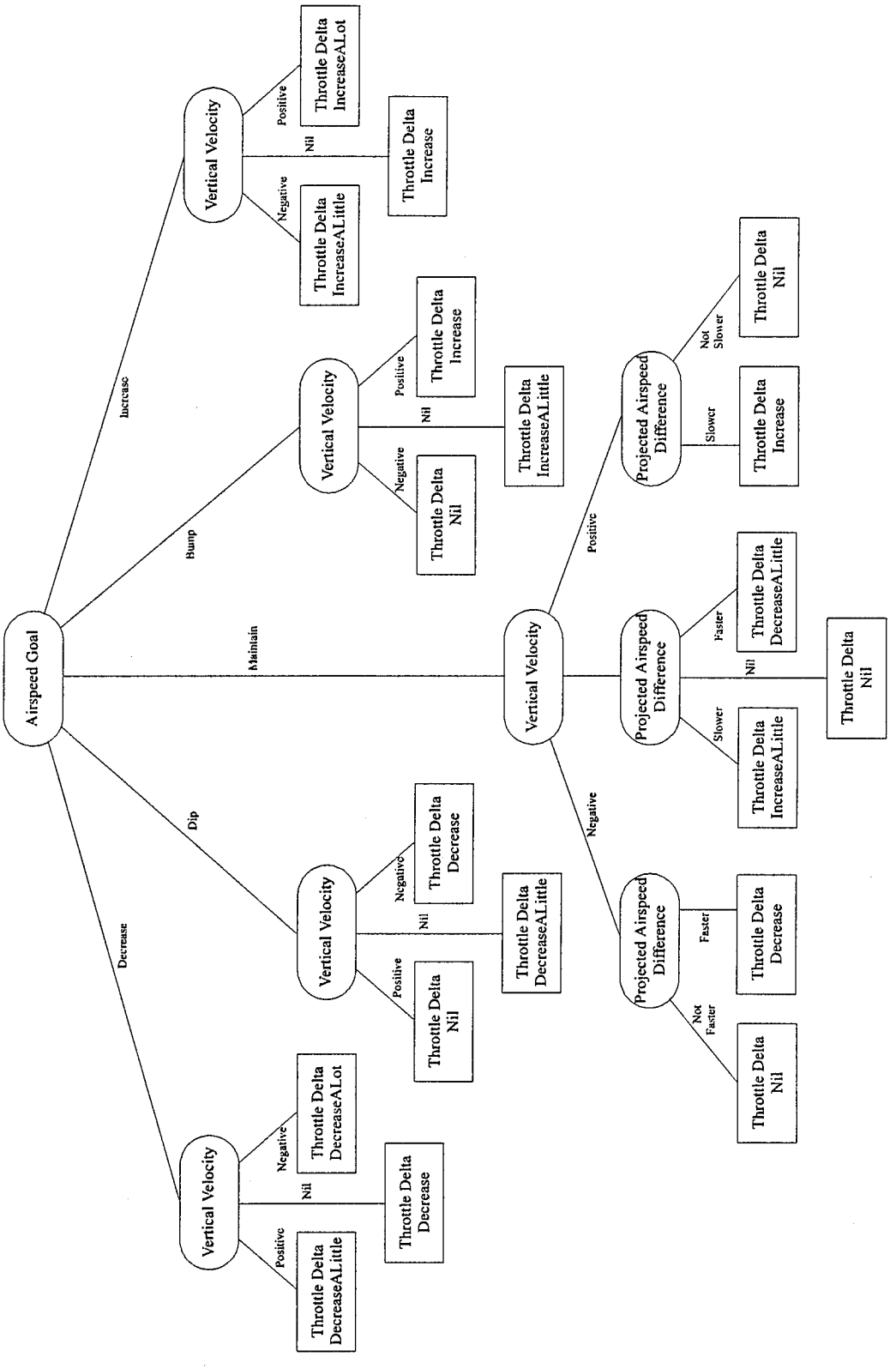

Figure 4-25 Throttle Delta Rule Graph

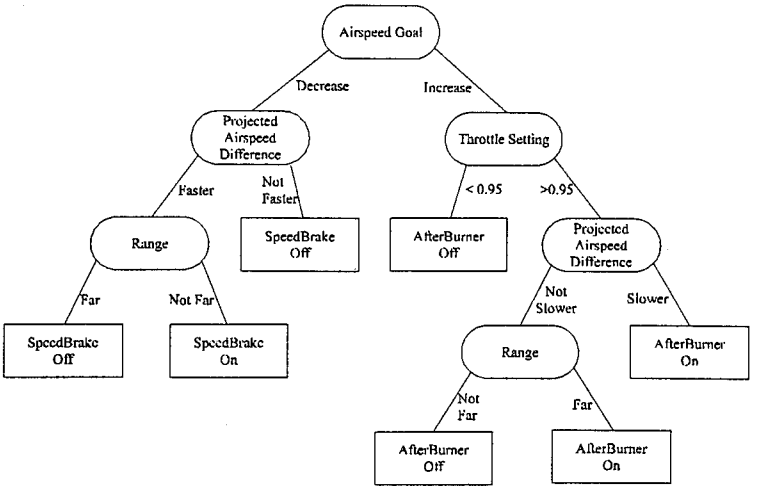

Figure 4-26 Afterburner and speedbrake Rule Graph

4.5 Voice Command Enumeration

The Automated Wingman is a semi-automated force and will accept control commands from the pilot of the lead simulator. However, the Automated Wingman cannot interpret all English language commands. Therefore, a command set must be developed to provide a set interface between the pilot and the Automated Wingman. This section describes that interface, including the command structure and vocabulary.

In keeping with true military style, I intend to use a two step command process. The first step is called the preparatory command and the second is the action command. The preparatory command conveys all the information that the subordinate needs to know about the command while the action command instructs him to execute. Consider the command "Forward, March!". The command "Forward" is the preparatory command. It tells subordinates that they are about to move forward. They do not execute at this time. Executing now would lead to haphazard starts and uncoordinated movement. Instead, the subordinates wait for the execution portion of the command. This provides a common,

coordinated point at which to begin the execution of the command. The same will be true of the commands used to control the Automated Wingman. I have chosen this command structure to allow for commands of variable length. This way, the leader can convey all the information about the command to the wingman before the wingman goes out and executes any portion of the command. This gives the leader the opportunity to correct himself or change the command before execution.

In order to understand how this structure will work it is necessary to know the command vocabulary. Table 4-14 and Table 4-15 describe the enumeration of voice commands to be used in the Automated Wingman project.

Table 4-14 Enumerated Preparatory Commands

Enumerate Maneuvers		Enumerated Modifiers	
010	Dive	130	Steep
011	Climb	131	Shallow
012	Bank	132	Hard
013	Roll	133	Easy
014	Altitude Flight Level	134	Fast
015	Airspeed | Speed	135	Slow
016	Heading	136	Tight | Tightly
017	Orbit	137	Loose | Loosely
018	Strafe	Enumerated Target Designators	
019	Engage	200	Bearing
020	Evade	201	Designate
021	Slot Formation	202	Aircraft | Fighter | Bomber | Target
022	Wedge Formation	210	Bogey1
Enumerated Directions		211	Bogey2
100	North	212	Bogey3
101	Northwest	213	Bogey4
102	East	214	Bogey5
103	Southeast	215	Bogey6
104	South	216	Bogey7
105	South West	217	Bogey7
106	West	218	Bogey8
107	Northwest	219	Bogey9
108	Up		
109	Down		
110	Right | Starboard		
111	Left | Port | Port-side		

Table 4-15 Enumerated Action Commands

001	Break | Now | Go
255	Reset

Numbers will be transmitted as numbers.

Using the command structure described above and the vocabulary note that we can support complex commands such as

"Bank Hard Right, Now"
"Climb to Flight Level 250, Break" (Note: FL250 is Altitude 25000 ft)
"Designate Target Bearing 037, Flight Level 300 as Bogey1, Break"
"Bank Right, No, Reset, Reset, Evade, Go!"

This will provide the Automated Wingman with a robust capability to support a wide variety of commands.

4.6 Coordinate Systems

The last topic to be covered in this chapter is coordinate systems. The Automated Wingman contains four separate, but related, coordinate system, in which the motion of entities can be described (see Figure 4-27). This section discusses those coordinate systems and the design of the objects that converts between them.

4.6.1 DIS Coordinates

The DIS standard calls for all entity information to be specified in a round earth coordinate system called WGS-84. In the WGS-84 coordinate system the origin is at the center of the earth, the x-axis passes through the prime meridian at the equator, the y-axis passes through 90° east longitude at the equator, and the z-axis through the north pole [IEEE94]. This coordinate system allows an entity to be placed anywhere in the world and can support large operations. It is a universal coordinate system based on standards already in place. Therefore, it was selected as the DIS standard coordinate system.

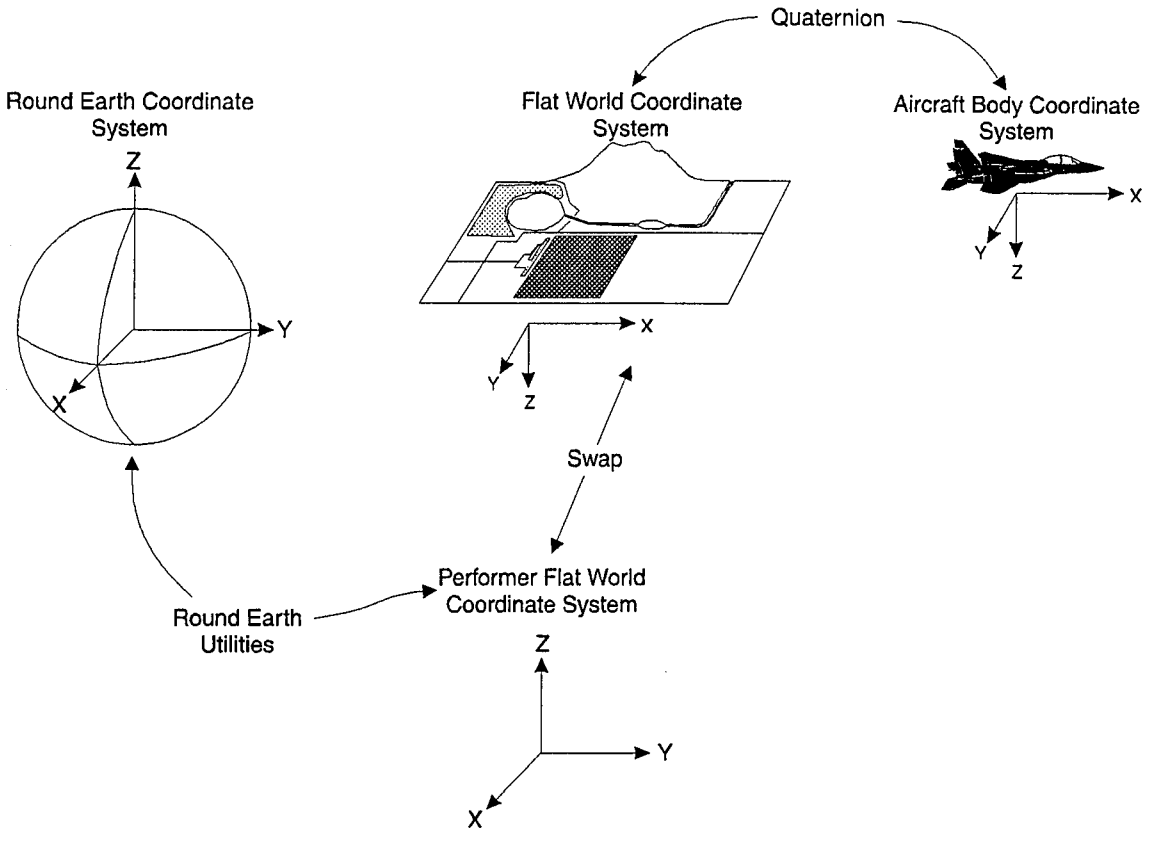

Figure 4-27 Coordinate Systems

4.6.2 Flat Earth Systems

Round earth models, such as WGS-84, are very convenient for locating battlefields all around the earth, but make for very large-valued coordinates once the battlefield has been placed. This large-valued coordinate system may cause a loss of precision in the local area of the battle. For that reason, WGS-84 is often not used within simulators. Instead, a local flat earth coordinate system is defined with the origin at a point specified in round earth coordinates. Angles are then used to measure the axes alignment of the two coordinate systems. Since the origin of the new coordinate system is nearby, coordinate values are not large enough make precision an issue. However, conversions are necessary in order to make these flat earth entities DIS compliant.

There are two possible orientations of a flat earth coordinate system, called NED and ENV. NED coordinate systems are those with their x-axis oriented to the north, y-axis to the east, and, by right-hand rule, their z-axis down. Hence, they are called NED for North, East, and Down. However, a more intuitive system is one in which the z-axis is positive in the up direction, so another orientation is sometimes used. In this alternative system, the x-axis points east, the y-axis points north, and the z-axis points vertical, or up. These are called ENV coordinate systems. Both ENV and NED can be used to perform the same function, but one must be careful to know which coordinate system is being used.

The Automated Wingman uses both systems. This is an artifact of the conversion of the VC aerodynamics model and supporting utilities. The aerodynamics model assumes an NED flat earth coordinate system while SGI's Performer library requires an ENV one. Therefore, both exist in the VC and the Automated Wingman, requiring care to be exercised when dealing with coordinate systems. At some point, the ENV coordinate system should be removed from the Automated Wingman, eliminating some of the troubling coordinate system issues in this project.

4.6.3 Aircraft Body Coordinates

The last coordinate system used in the Automated Wingman is the Aircraft Body Coordinate (ABC) system. The ABC system defines the x-axis to be pointing out the nose, the y-axis out the right wing, and the z-axis out the bottom. This is always true, regardless of the orientation of the airplane within the environment. A set of angles, called euler angles, are used to describe the orientation of the ABC system within the flat earth

system. Using these euler angles, any coordinate within the ABC can be transformed into the flat earth coordinate system.

4.6.4 Conversion Objects

The Automated Wingman uses two objects to make conversions between the various different coordinate systems. The first, borrowed from the AFIT VC, is called the RoundEarthUtilities object. It converts coordinates, vectors, and euler angles between the WGS-84 coordinate system and the flat earth ENV coordinate system. Although this object was extensively re-implemented to eliminate machine dependence, the basic design was not changed in order to maintain as much commonality between the VC and the Automated Wingman as possible. Thus, the ENV coordinate system was incorporated into the Automated Wingman. The RoundEarthUtilities object is capable of converting in both directions and is used extensively throughout the Automated Wingman.

The second object, the quaternion object, is used to transform the ABC system into flat earth NED. NED is then converted to ENV by swapping x and y values and negating z. Quaternions provide an efficient mechanism for incrementally updating a rotation matrix [SHOE93] which converts ABC system coordinates to flat earth NED. Hence, Cooke used them to convert the output of his aerodynamics model into flat earth NED coordinates [COOK89]. Therefore, the Automated Wingman uses the quaternion object to make conversions between the aircraft body and flat earth NED coordinate systems.

4.6.5 Summary

Figure 4-27 summarizes the coordinate systems used in the Automated Wingman and shows how they are converted. A future redesign would be to create an object that converts flat earth NED to round earth without first converting to flat earth ENV. The redesign would save precious clock cycles and make the system more comprehensible.

4.7 Conclusion

This chapter has examined the design issues involved in developing the Automated Wingman. It covered the basic architecture and showed how it is designed to provide an interface between the DIS objects, the Airplane object and the FuzzyPilot object. Next, the two main objects of the Automated Wingman were examined. The Airplane object provides the flight dynamics and capabilities of an airplane while the FuzzyPilot provides the intelligence that imitates a human pilot. Then, the design of the fuzzy expert system was explored. The knowledgebase hierarchy and several of the linguistic variable designs were shown to demonstrate how the FuzzyPilot uses information to determine what action should be taken next. Furthermore, the design of the voice command set was shown. This included an enumeration of commands and explained how they are to be used. Finally, a discussion of the various coordinate systems used by the Automated Wingman was presented. All these concepts have been brought together in the Automated Wingman to create a realistic computer generated semi-automated entity for use in distributed simulations.

5. Implementation

5.1 Introduction

This chapter discusses the details of the implementation of the Automated Wingman. I begin with an explanation for choosing C++ as the implementation language. Next, I detail some of the classes that I developed to hold key data components, called container classes. Then, I discuss the rationale for choosing FuzzyCLIPS as the fuzzy expert system shell, as well as the interface between the Automated Wingman and FuzzyCLIPS. Particular interest paid to data structure issues between C++ and FuzzyCLIPS. Following the discussion of FuzzyCLIPS, I present some examples of how the linguistic variable and production rule designs from Chapter 4 are transformed into FuzzyCLIPS constructs. Finally, I conclude this chapter with an overall assessment of the implementation and indicate some of the areas left for further development.

5.2 Language Considerations

When it came to selecting a language for implementing the Automated Wingman, there were basically two choices, C++ and Ada. Both are object oriented and support the class structure shown in Figure 4-3. Ada is the DoD standard language and is gaining widespread support. However, C++ is an industry standard and, as such, is supported by many companies selling compilers and other development tools. C++ is also favored for research because of its flexibility and the fact that it interfaces naturally with the Unix operating system used on the class of workstations for which the Automated Wingman is

designed. Add the fact that nearly all of the fuzzy inferencing tools I found were C or C++ based and the fact that the Object Manager DIS interface is written in C++, then C++ became the obvious choice for the implementation language. Choosing Ada or any other language would have required a tremendous effort to duplicate the tools that already exist in C++. Therefore, I chose to implement the Automated Wingman in the C++ programming language.

5.3 Container Classes

In the design phase the software analyst assumes the existence of data structures that, in practice, may not actually exist. This abstraction helps the analyst to see the "big picture" and not get bogged down in the details of a particular programming paradigm. However, when the programmer encounters these data structures, he must decide how to implement them without departing from the design. In object-oriented programming these data structures are usually made into objects called *container classes* [RUMB91]. These container classes form the basis of the rest of the implementation.

The Automated Wingman uses three container classes, shown in Figure 5-1. The *Matrix* class has two subclasses, *HMatrix3D* and *HVector3D*. The *Matrix* and *Quaternion* classes each hold data to be operated on and are therefore implementations of the data structures used in Chapter 4. *RoundEarthUtilities*, on the other hand, is a container class for coordinate system conversions. This section provides the details of these classes.

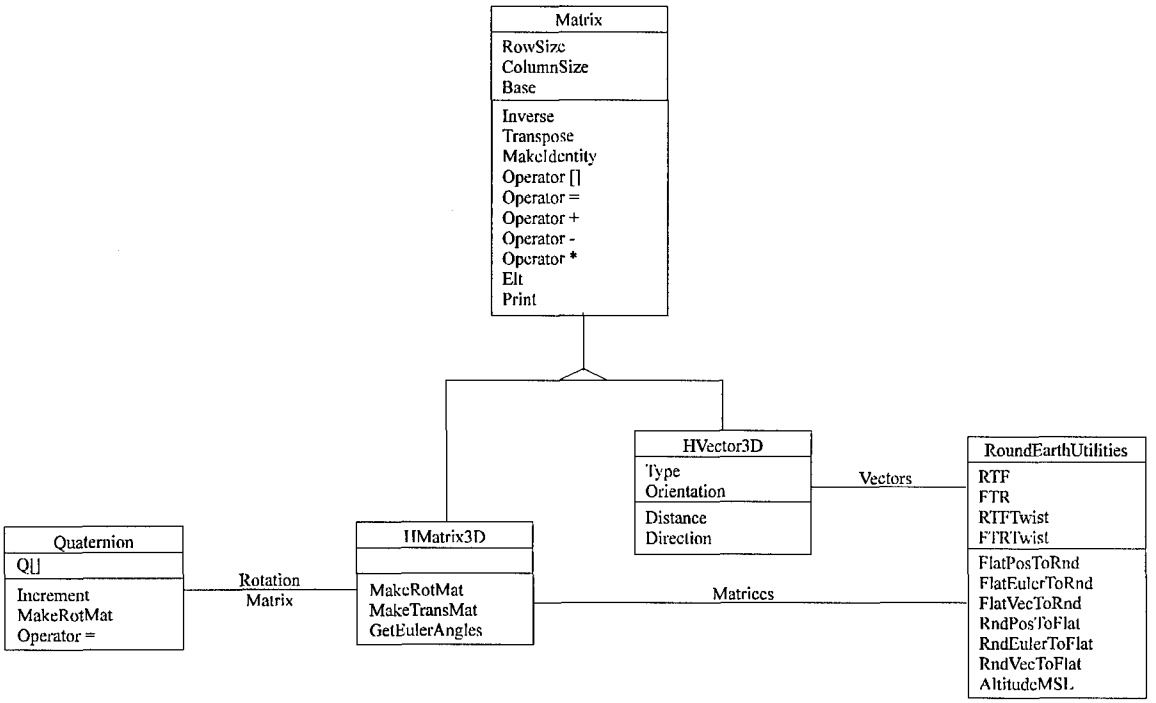

Figure 5-1 Container Classes Used in the Automated Wingman

One feature that each of these container classes have in common is that they have been developed using templates. Templates are a C++ construct that allow the developer to abstract away the specific data type, such as **int** or **float**, and deal with just the operations themselves. Accordingly, the programmer creates a template of the class that tells the compiler how the data objects are to be manipulated. When the programmer wants to use the class, he instantiates it with a specific data type. The compiler then creates the actual code, complete with types, based on the class template provided by the programmer. Templates are a powerful and convenient way to implement container classes that perform the same functions without regard to the type of data being stored. I have therefore used them in the implementation of the container classes for the Automated Wingman.

5.3.1 The Matrix Class

The *Matrix* class provides a data structure for holding matrices of various sizes, controlled by the attributes *RowSize* and *ColumnSize*. There are several methods, such as Transpose and Inverse, and operators, like "+, "-", and "[]", designed to allow the matrices to be manipulated. Although the class was originally created by Capt Jim Russell, I have made extensive modifications to eliminate compiler warnings and to improve functionality. Once I had the *Matrix* class, I was able to proceed to the container classes that the Automated Wingman really requires.

The first subclass, *HMatrix3D*, implements a three-dimensional homogenous matrix. A three dimensional homogenous matrix is a 4 x 4 matrix used for rotating and transforming coordinate systems. Since the Automated Wingman uses several coordinate systems, it requires the ability to store matrices that can perform coordinate system transformations. This class includes methods to generate rotation and translation matrices as well as a method to extract euler angles from a direction cosine matrix. It also inherits all the methods of the Matrix base class, although some are overloaded (not shown in the figure) to take advantage of the fact that an *HMatrix3D* object is of known size. The primary difference between a *Matrix* object and an *HMatrix3D* object is that the *HMatrix3D* object is constrained to be a 4 x 4 matrix while the *Matrix* object can be of any dimension.

The second subclass is the *HVector3D* class. An *HVector3D* object is either a 4 x 1 column vector or a 1 x 4 row vector. The orientation is indicated by the attribute "Orientation", which can take on the value "Row" of "Column". An *HVector3D* can

describe either a point in 3-space or a vector as indicated by the "Type" attribute. If the object is a point then the last element is 0, otherwise, it is 1. Like *HMatrix3D*, *HVector3D* inherits all the attributes and methods of the *Matrix* class. It also adds methods used with vectors such as distance, which is the vector distance between two vector objects, and direction, which returns a unit vector in the direction of the object. *HVector3D* provides the functions necessary to manipulate points and vectors within the various coordinate systems used in the Automated Wingman.

5.3.2 The Quaternion Class

The purpose of a quaternion was described in Chapter 4. Figure 5-1 shows how the quaternion class is implemented. The quaternion itself is stored as a four element array. The first method incrementally updates the quaternion with the changes in angular velocity of the Automated Wingman. The second method generates a rotation matrix from the quaternion. The last method allows the programmer to set two quaternions equal to each other and is provided for the cockpit switching capability implemented by Schneider in [SCHN95]. Like the *Matrix* class, the *Quaternion* class has been implemented as a template.

5.3.3 The RoundEarthUtilities Class

The last container class designed for the Automated Wingman does not store data. Instead, it implements conversions between the WGS-84 round earth coordinate system and the ENV flat earth coordinate system (see chapter 4). The class is called *RoundEarthUtilities*. It is based on a class by the same name developed by Gerhard and Erichsen [ERIC93]. However, that class used matrix and vector functions from the SGI

Performer libraries to create and manipulate coordinate system transformations. The implementation developed for the Automated Wingman eliminates these dependencies, using the *HMatrix3D* and *HVector3D* classes instead. An excellent description of the original class can be found in [ERIC93]. Since the Automated Wingman's *RoundEarthUtilities* class represents a re-implementation of the original design, the reader is referred to [ERIC93] for further information.

5.4 FuzzyCLIPS

This section discusses FuzzyCLIPS, the fuzzy expert system shell chosen for this project. The rationale for choosing FuzzyCLIPS is discussed, as well as some implementation objects that organize the data and help with the C++/FuzzyCLIPS interface. Although FuzzyCLIPS is the best choice available for this project, there were still some issues that had to be dealt with. The first of these is the fact that the Automated Wingman is written in C++ and FuzzyCLIPS is written in C. Second, FuzzyCLIPS was designed as a standalone program. A programming interface was added to improve the functionality of FuzzyCLIPS, but it is not easy to use. Both of these required special attention in the implementation of the Automated Wingman.

5.4.1 Rationale

FuzzyCLIPS was chosen as the fuzzy expert system shell for several reasons. The first reason is that it is based on a widely known expert system shell called CLIPS. CLIPS is a project of NASA that has been ongoing for many years. There is a great deal of information available about CLIPS and its capabilities, as well as its limitations, are well

understood. As a long-term project, CLIPS is reasonably mature and it includes many advanced features, such as Rete pattern matching [GIAR94], not found in some other implementations of expert system shells. FuzzyCLIPS is an extension of CLIPS. It maintains all of the capabilities of the original, non-fuzzy, version but it adds the capability to represent fuzzy concepts. Hence, all the literature about CLIPS applies to FuzzyCLIPS.

The second reason that FuzzyCLIPS was chosen is that, like CLIPS, it is C based and has a C language interface. Since I had already selected C++ as my development language, choosing a C based system made sense. There are other fuzzy expert system shells written in C and C++; however, most lack either the sophistication or the capabilities provided by FuzzyCLIPS. Therefore, FuzzyCLIPS was an appropriate choice.

Finally, FuzzyCLIPS is available free of charge. That meant that I could easily obtain it and begin development immediately. For these three reasons, FuzzyCLIPS became the fuzzy expert system shell for the Automated Wingman.

5.4.2 Implementation Objects

FuzzyCLIPS supports an object-oriented language called COOL (CLIPS Object-Oriented Language). COOL allowed me to create objects within FuzzyCLIPS that helped organize the data and streamline the C++/CLIPS interface. The objects I created are described in this section.

5.4.2.1 The Airplane Object

The FuzzyCLIPS Airplane object is an abstract object with two concrete subclasses, Wingman and Lead. The object model is shown in Figure 5-2. When placed

on the FuzzyCLIPS fact list, this structure holds information about both the Automated Wingman and the lead simulator. Modules can access this information in order to apply it in the reasoning process. This information is updated every time the decision loop is entered.

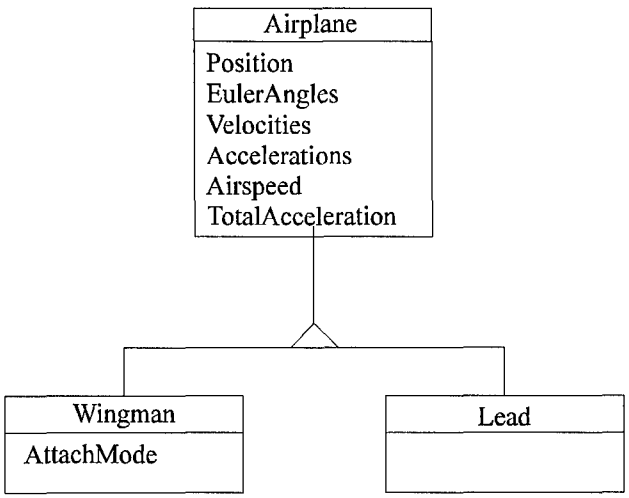

Figure 5-2 Airplane Object within CLIPS

5.4.2.2 The FlightControl Object

In addition to accessing the Wingman and Lead objects shown in Figure 5-2, the FlightControls module also has a private object, the FlightControls object. The FlightControls object, shown in Figure 5-3, contains the current setting for the TAS, afterburner and speedbrake. The FuzzyCLIPS FlightControls module stores the new control information in this object, while the C++ FlightControl class accesses it to get the

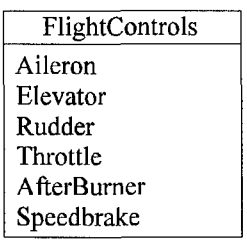

Figure 5-3 The FlightControls Object

new TAS settings from FuzzyCLIPS. Therefore, this object acts as an interface between the expert system and the underlying architecture of the Automated Wingman.

5.4.3 Language Issues

As mentioned above, FuzzyCLIPS is a standalone program written in C. This means that it is not object-oriented nor is it a library. In order to imbed FuzzyCLIPS into the Automated Wingman, I had to recompile and link FuzzyCLIPS as a library. CLIPS was designed to do this and the fuzzy extensions in FuzzyCLIPS do nothing to interfere with this. However, there are several switches in the "setup.h" file that had to be changed [JSC93b] as well as modifying the Makefile that came with FuzzyCLIPS. I compiled FuzzyCLIPS into a library called "libFzCLIPS.a". This file, and the Makefile that built it, can be found in "~medwards/thesis/fzclips" on the SGI systems.

FuzzyCLIPS provides a header file, called "clips.h", that declares all of the functions found in the user interface. This had to be included in the compilation units that required access to FuzzyCLIPS. Two problems arose from this. The first was with naming conventions. FuzzyCLIPS does not use distinctive variable names that other programmers are not likely to use (such as appending _CLIPS or a couple of underscores), so, a few name collisions occurred. These were solved by ensuring that the colliding types were never used in the same compilation unit. The second problem was with interfacing C with C++. In order to support method overloading, C++ uses a process called name-mangling, which adds characters to the method names of a class. These characters are based on the function's formal parameter list. Thus, two functions with identical names but different parameters are represented differently in C++. However, C

does not support name-mangling. Therefore, the C++ compiler must be informed when a particular section of code that is not name-mangled is included in a C++ source file. This is done with the 'extern "C" {}' construct. Whenever the "clips.h" file is included, it is wrapped in this construct so that the C++ compiler can work with it.

5.4.4 Programming Interface

The FuzzyCLIPS programming interface, which is based on the CLIPS interface in [JSC93b], is very cumbersome. This is because CLIPS treats all data as objects, even numbers and characters. For example, if a CLIPS variable is set to the value "3", a "3" object is created, placed in memory, and the variable set to point at it. Other CLIPS variables with the value "3" will also point to that object. This is efficient if large numbers of variables are set to "3", and is particularly useful for speeding up the pattern matching used by the Rete algorithm [GIAR94]. However, this means that CLIPS manages its own memory and that the data structures used by CLIPS are non-standard. Therefore, moving data into CLIPS, hence FuzzyCLIPS, requires the use of special functions. Additionally, it means that data in the Automated Wingman must be "loaded" into FuzzyCLIPS and the results must be "read out" from FuzzyCLIPS. These functions are provided by the programming interface but are difficult to use.

The Automated Wingman uses FuzzyCLIPS objects to transfer data between the basic architecture and the FuzzyCLIPS fact list. This is because it is difficult for the basic architecture to find the address of an asserted FuzzyCLIPS fact, such as the result of a reasoning cycle. Objects, on the other hand, can be created by the basic architecture, which stores the object's memory address, and accessed whenever needed. This allows

the basic architecture to place data into FuzzyCLIPS objects, instruct FuzzyCLIPS to "run", and then read the result from FuzzyCLIPS as long the rulebase puts the result back into an object. This is precisely how the Automated Wingman interface to FuzzyCLIPS was implemented. The Automated Wingman, through the FlightControls class, creates three objects: the Wingman object, the Lead object (see fig 4-10), and the FlightControls object (see fig 4-11). Data from the Automated Wingman is placed into the first two of these objects. Other data, such as the lookahead period used for the linguistic variables projected into the future, are also placed on the FuzzyCLIPS fact list. The FlightControls class then issues the FuzzyCLIPS "run" command. When the "run" command completes, the rulebase has placed the new TAS settings into the FuzzyCLIPS FlightControls object. The Automated Wingman reads these values out, places them into the Control class, and then executes the Aeromodel with the new TAS settings as input. Clearly, it is a cumbersome interface. However, with a good deal of effort, it has been made to work for the Automated Wingman.

5.5 FlightControl Knowledgebase

Chapter 4 presented the design of the FlightControl knowledgebase. All of the linguistic variable designs were shown in tables and figures, as well as the design of the production rules. However, these must be transformed into FuzzyCLIPS constructs at implementation time. This section explains how that is accomplished.

FuzzyCLIPS supports a construct called *deftemplates* in which linguistic variables can be represented. The general structure of a fuzzy deftemplate is:

```
(deftemplate <TemplateName> ["comment"]
    <from> <to> [<units>] ; universe of discourse
    (
        (<TermSetName> <description of fuzzy set>)
        .
        .
    )
)
```

where:

<TemplateName> is the name of the linguistic variable
<from> is the start of the interval over which the linguistic variable is defined
<to> is the end of the interval over which the linguistic variable is defined
<units> is the units of the interval [optional]
<TermSetName> is the name of the term set being defined
<description of fuzzy set> is a fuzzy set specification (see [KSL94])

For example, the FuzzyCLIPS implementation of the CurrentRelativeAltitude linguistic variable is:

```
(deftemplate CurrentRelativeAltitude
    -20000  20000  Meters
    (
        (Lower    (-50 1)(-30 0))
        (Nil      (-50 0)( 0 1)(50 0))
        (Higher   ( 30 0)( 50 1))
    )
)
```

Similarly, the production rule graphs from chapter 4 must be implemented. The format of a FuzzyCLIPS rule is given in [KSL94] and is very similar to the format of a CLIPS rule [GIAR94] [JSC93a]. The only difference is the inclusion of Certainty Factors in FuzzyCLIPS rules, which are not used in this implementation of the Automated Wingman. As an example, a rule derived from the Vertical Velocity Rule Graph (Figure 4-22) is shown below.

```
(defrule InThePocket
    (object  (is-a                    WINGMAN)
             (AttachMode              Attached))
    (CurrentRelativeAltitude          Nil)
    (VerticalVelocityDifference       Nil)
    (ProjectedRelativeAltitude        Nil)
    =>
    (assert (VerticalVelocityGoal     Maintain))
)
```

This rule is developed by following path in Figure 4-22 that goes straight down the middle of the figure. From these two examples, it is easy to see that the implementation of the design presented in Chapter 4 is straight-forward.

5.6 Future Work

This is the initial version of the Automated Wingman. Accordingly, a great deal effort was put into creating a design that could be extended and added to without major modifications to the underlying structure. I attempted to identify, separate, and modularize the functions required to fly an airplane. I believe that I was successful in this endeavor. I also implemented basic flight control using fuzzy logic. It is a low level of implementation, but it works. However, the Automated Wingman must be developed much further before it can be included in a distributed simulation.

This version of the Automated Wingman implements a basic flight control package using fuzzy logic. These flight controls can fly the airplane along a route given by a series of points, called "fly-to-point" operation. However, this is the lowest level of control and works best when the Automated Wingman is attempting to fly in formation with a lead aircraft simulator. Higher levels of control are needed to implement more complex maneuvers and even higher levels to implement tactics. In addition, the current version of

the Automated Wingman has no weapons or ability to reason about what weapons should be employed. For the Automated Wingman to be useful in DIS simulations, weapons, maneuvers, and tactics must be added.

5.7 Conclusion

This chapter has discussed some of the details of the implementation of the Automated Wingman. C++ was chosen as the implementation language in order to re-use as much already written code as possible. The implementation details of the linguistic variables and rule graphs shown in chapter 4 are presented along with representative examples of each. Finally, two issues involved in integrating FuzzyCLIPS were discussed. These are creating the FuzzyCLIPS library to embed into the Automated Wingman and a discussion of the cumbersome, but functional, programming interface provided by FuzzyCLIPS. Finally, suggestions for future work are presented. These are the significant issues encountered in the implementation of the Automated Wingman.

6. Results/Conclusions

This chapter describes the results of the Automated Wingman project. Particular emphasis is placed on the fact that the fundamental design of the Automated Wingman is itself a result. Also, the results of testing the flight control module are presented. First, the design of the Automated Wingman is discussed. Next, data is presented to demonstrate the effectiveness of the flight control module. From these results, I draw conclusions about the project. Finally, I make some recommendations about work that must be done before the Automated Wingman can be a viable entity in a distributed simulation.

6.1 Results

This section describes the results obtained from this project. Specifically, it discusses the flexibility and extendibility of the fundamental design and the results obtained from the flight control module.

6.1.1 The Design

One of the results of the Automated Wingman is the fundamental design. My goal was to achieve a flexible and extendible design that could serve the basis for future work on the Automated Wingman. With the design presented in chapter 4, I have met this goal. The flexibility of the Automated Wingman is a result of decomposing the problem into two distinct parts, the FuzzyPilot and the Airplane. The expandability is provided by the

clearly defined class interfaces and modular code, as well as by the choice of FuzzyCLIPS as the reasoning tool. Both of these aspects are explained in this section.

The decomposition of the Automated Wingman into the FuzzyPilot and Airplane objects maintains the separation between the distinct parts of the problem. This separation allows the developer to view the problem in real world terms and work with domains that are more familiar and in which experts exist. These domains are aircraft design and capabilities, and flying a high performance jet fighter, which can, in turn, be decomposed further into the domains presented in chapter 4. In the aircraft design domain, work is already underway to replace the existing aerodynamics model with a better one. The goal is to eventually have a library of different aerodynamics models, each representing a different type of aircraft that can selected when the Automated Wingman is instantiated. The weapons store is another example of an object that can be added to the aircraft domain. The class structure devised for the Automated Wingman will facilitate these efforts.

The fighter pilot domain is knowledge-based and uses the fuzzy expert system. It represents the "human" portion of the Automated Wingman. As of this version of the Automated Wingman, the pilot can fly his aircraft given a series of route points to fly along. However, the fighter pilot domain can be developed through further knowledge engineering efforts. This will result in the addition of more modules to the FuzzyPilot, which the structure is designed to accommodate. Therefore, knowledge can be added to the Automated Wingman without having to worry about improved aerodynamics models

or how the Airplane object does its job. The design removes this concern from the knowledge engineer making the Automated Wingman flexible.

The expandability of the Automated Wingman is also a result of the design. Since the Automated Wingman is object-oriented, new objects can be identified and added without restructuring the entire system. Most existing classes have methods allowing internal data to be read. If required, additional methods can be added to any class without impacting the overall structure. This expandability is essential if the Automated Wingman is to fulfill the role of a believable semi-automated force.

FuzzyCLIPS also enhances the expandability of the Automated Wingman. Knowledgebases can be modularized within FuzzyCLIPS to maintain their separation. Therefore, it is possible, and desirable, to create new knowledgebase modules as new reasoning capability is added. This modularity improves efficiency, eliminates collisions between knowledgebases, and allows unrelated knowledgebases to be developed independently. Also, certain data elements identified as global in scope can be made available to all FuzzyCLIPS knowledgebase modules. This flexibility in knowledgebase design and implementation greatly enhances the expandability of the FuzzyPilot and the Automated Wingman as a whole.

6.1.2 Flight Control Results

Three axes of control were identified for implementation in the flight control module, elevation, airspeed, and heading. These were decoupled as much as possible, although airspeed and heading are related, so a complete separation was not possible. Next, knowledgebases were designed and implemented to control the motion of the

Automated Wingman along these axes. Then, these knowledge bases were tested and the results graphed. This section presents those results.

The Automated Wingman was tested by using it as a wingman to an aircraft created by the AFIT Gaggle generator and then tracking the performance of the Automated Wingman over time in each of the three axes. The AFIT Gaggle generator is a simple program whose purpose is to generate and broadcast DIS entities. Each entity in the Gaggle is programmed to fly a particular route over the terrain. To test the Automated Wingman, I devised a race-track style route where the Gaggle entity flies straight for nearly the length of the terrain, turns, and then flies back. This repeats until the Gaggle is terminated. With the Gaggle entity designated as the lead simulator, I ran the Automated Wingman, which was programmed to output the time, its relative altitude, airspeed, and heading. Representative results from these runs are presented in this section.

Figure 6-1 shows the results in the elevation axis. The goal was to maintain the

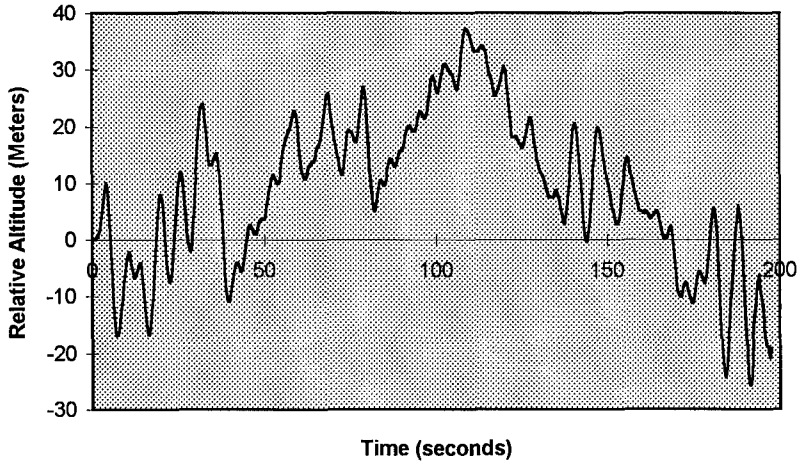

Figure 6-1 Vertical Motion Analysis

relative altitude within ±50 meters and the graph shows that this goal was satisfied. The jaggedness of the graph is attributed to oscillations in the aerodynamics model due to coordinate conversions. Smoothing the current model or implementing a new model altogether will be required to remove these oscillations.

6.2 Conclusions

The original goals of the Automated Wingman were lofty. They were based on assumptions about the re-usability of code from the AFIT Virtual Cockpit and the availability of tools for the implementation of the voice control. Unfortunately, these assumptions proved to be, at best, exaggerated and the implementation is not as robust as originally intended. Still, this thesis has demonstrated the feasibility of a fuzzy logic based semi-automated force. I have created a fundamental design that is flexible and ready to serve as the foundation of future efforts on this project. My implementation has shown that a hierarchy of fuzzy linguistic variables can be used to control a dynamic process, such as an airplane. Therefore, I conclude that this thesis has successfully proven the concept of a semi-automated force based on fuzzy logic.

This thesis has laid the groundwork for the Automated Wingman. It has provided an underlying design structure that is open, flexible, and extendible. I have succeeded in creating a design where the capabilities that can be added to the Automated Wingman are limited only by imagination of the developer and processing power of available computers. As more students develop the reasoning hierarchy, I have every reason to expect that the Automated Wingman will successfully participate in distributed simulations and enhance their training value.

6.3 Recommendations

The Automated Wingman has tremendous potential to provide believable computer generated semi-automated forces for distributed simulations. By exploiting the capabilities of fuzzy logic, we can create simulation entities that attach themselves to other entities and are indistinguishable from manned simulators. This can help make distributed simulation a valid and cost-effective training tool. Therefore, development of the Automated Wingman should be continued.

This thesis has been successful in providing the fundamental design and flight control for the Automated Wingman. However, there is still much work to be done before the Automated Wingman can fulfill its mission. If this project is thought of as a pyramid, I have provided the base. The rest of the pyramid remains to be built. However, in my hierarchical design, I have provided a blueprint that can be followed, or modified as required, to fill-in the remaining portions of the Automated Wingman.

In order to fight in a distributed simulation, a weapons module must be added. The Automated Wingman must be given a weapons store and the ability to determine which weapon is appropriate for a given scenario. This implies that a weapons selection knowledgebase is required. This knowledgebase must contain the information on the capabilities of each available weapon, consider external factors such as the target type and weather, and be able to select the appropriate weapon for the given situation. Once the weapon is selected, the Automated Wingman must then know how to use it.

To employ the selected weapon, a tactics module must be developed. Tactics govern how weapon systems are used. For any selected weapon, the tactics may be

designed to position the aircraft in the correct place to launch the weapon or they may be guidance on how and when to release the weapon. Either way, a complex series of maneuvers will be required. The tactics knowledgebase must be capable of selecting the appropriate maneuvers to put the aircraft in a position to maximize the effectiveness of the chosen weapon. Therefore, a tactics knowledgebase needs to be added to make the weapons useful.

Tactics are implemented by a series of maneuvers. Hence, a maneuver module is required. This maneuver module can build upon the "fly-to-point" concept behind the implementation of the flight control module or a different module can be included to translate the maneuvers for use by the flight control module. This maneuver module must include knowledge and state, and must be goal oriented in order to know when a given maneuver is complete. The maneuver module is essential the for operation of the Automated Wingman and should be the next area developed in this project.

Another module that is important to the operation of the Automated Wingman is the voice interface. I had intended to implement this feature. However, the tools required to do this were not available when I needed them. However, they will eventually be ready. Once that happens, the Automated Wingman can be given the capability to receive and respond to voice commands based on the design given in chapter 4, as was intended in the original Automated Wingman concept.

A related feature is to have the Automated Wingman communicate with the lead simulator. One of the key responsibilities of a real wingman is to watch for threats and make recommendations to the lead. In order for the Automated Wingman to do this, it

must have its own message generation capability. The Automated Wingman could then track and analyze the environment and inform the lead when there is an imminent threat or a better option than the one currently being pursued. This realistic behavior would greatly enhance the believability of the Automated Wingman entity.

Each of these modules is necessary for the Automated Wingman to fulfill its requirements. This thesis provides a design that can easily incorporate these modules, as well as guide their development. However, in order to create a realistic SAF, this project must continue. Clearly, there is more work to be done on the Automated Wingman before it is ready to fly, fight, and win.

Bibliography

[BLAU94] Blau, B. and others, "The DIS (Distributed Interactive Simulation) Protocols and their Application to Virtual Environments," Institute for Simulation and Training, Orlando, FL, 1994.

[COOK92] Cooke, J., M., "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions," Presence, 1(4), 1992, 404-420.

[COX92] Cox, E., "Solving Problems with Fuzzy Logic," AI Expert, March 1992, 28-36.

[DIAZ94] Diaz, M., E., "The Photo-realistic AFIT Virtual Cockpit," Masters Thesis, AFIT/GCS/ENG/94D-02, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1994

[DINO91] Di Nola, A., and others, "Reduction Procedures for Rule-based Expert Systems as a Tool for Studies of Properties of Expert's Knowledge," *Fuzzy Expert Systems*, edited by A. Kandel, CRC Press, Boca Raton, FL, 1991.

[DUBO80] Dubois, D. and H. Prade, "Fuzzy Sets and Systems," Academic Press, New York, NY, 1980.

[ERIC93] Erichsen, M. N., "Weapon System Sensor Integration for a DIS-Compatible Virtual Cockpit," Masters Thesis, AFIT/GCS/ENG/93-07, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1993.

[FORT94] Fortner, J., L., "Distributed Interactive Simulation Virtual Cassette Recorder (DIS VCR): A Datalogger with Variable-Speed Relay," Masters Thesis, AFIT/GCS/ENG/94-XX, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1994.

[GERH93] Gerhard, W., E., "Weapon System Integration for the AFIT Virtual Cockpit," Masters Thesis, AFIT/GCS/ENG/93-XX, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1993.

[GIAR94] Giarratano, J. and Riley, G., *Expert Systems: Principles and Programming*, PWS Kent, Boston, 1994.

[GOSS94] Gossweiler and others, "An Introductory Tutorial for Developing Multi-user Virtual Environments," Presence, 3(4), 1994, 255-264

[HADD93] Haddix, R., G., "An Immersive Synthetic Environment for Observation and Interaction with a Large Volume of Interest," Masters Thesis, AFIT/GCS/ENG/93-XX, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1993.

[HALL91] Hall, L., O., and A. Kandel, "The Evolution From Expert Systems to Fuzzy Expert Systems," *Fuzzy Expert Systems*, edited by A. Kandel, CRC Press, Boca Raton, FL, 1991.

[IST94] "Standard for Distributed Interactive Simulation - Application Protocols, Version 2.0 Fourth Draft," Institute for Simulation and Training, Contract #N661339-91-C-0091, Orlando, FL, 1994.

[JSC93a] Johnson Space Center, CLIPS Users Guide, National Aeronautics and Space Administration, Cape Canaveral, FL., 1993.

[JSC93b] Johnson Space Center, CLIPS Advance Programming Guide, National Aeronautics and Space Administration, Cape Canaveral, FL, 1993.

[KAND86] Kandel, A., "Fuzzy Mathematical Techniques with Applications," Addison-Wesley, Reading, MA, 1986.

[KLIR95] Klir, G., J., and Yuan, B., *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall PTR, Upper Saddle River, NJ, 1995.

[KOSK92a] Kosko, B., "Chapter 7 - Fuzziness Vs Probability," *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[KOSK92b] Kosko, B., "Chapter 8 - Fuzzy Associative Memories," *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[KOSK93] Kosko, B., *Fuzzy Thinking: The New Science of Fuzzy Logic*, Hyperion, New York, NY, 1993.

[KUNZ94] Kunz, A., A., and Stytz, M., R., "A Virtual Environment for Satellite Modeling and Orbital Analysis in a Distributed Interactive Simulation," Proceedings of the Military, Government and Aerospace Simulation Conference, San Diego, CA., 1994, 55-60.

[KSL94] Knowledge Systems Laboratory, *FuzzyCLIPS Version 6.02A User's Guide*, Institute for Information Technology, National Research Council Canada, Ottawa, Ontario, Canada, Sep 1994.

[LAIR95] Laird, J., E. and others, "Simulated Intelligent Forces for Air: The Soar/IFOR Project 1995," Proceedings of the Fifth Conference on

Computer Generated Forces and Behavioral Representation, Orlando, FL, May, 1995.

[MCCA94] McCarty, W. D., and others, "A Virtual Cockpit for a Distributed Interactive Simulation," *IEEE Computer Graphics and Applications,* Jan 1994, 49-54.

[MYER86] Myers, W., "Introduction to Expert Systems," IEEE Expert, 1(1), Spring 86, 100-109.

[MUNA94], Munakata, T. and Y. Jani, "Fuzzy Systems: An Overview," Communications of the ACM, 37(3), March 1994, 69-76.

[NEAP91] Neapolitan, R. E., "A Survey of Uncertain and Approximate Inference," in *Fuzzy Logic for the Management of Uncertainty*, edited by L. A. Zadeh and J. Kacprzyk, Wiley, New York, 1992.

[NEGO84] Negoita, C., "Chapter 1 - Introduction," *Expert Systems and Fuzzy Systems*, Benjamin Cummings, Menlo Park, CA, 1984.

[ROGE94], Rogers, D., "STOW-E Lessons Learned - Focused on the 3Primary Army STOW-E Sites", Cubic Defense Systems, February, 1995.

[ROSE91] Rosenbloom, P., S. and others, "A Preliminary Analysis of the Soar Architecture as a basis for General Intelligence," Artificial Intelligence, 47, 1991, 289-325.

[ROSE93] Rosenbloom, P., S. and others, *The Soar Papers: Research on Integrated Intelligence*, MIT Press, Cambridge, MA, 1993.

[RUMB91] Rumbaugh, J., and others, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[SCHN91] Schneider, M., and A. Kandel, "General Purpose Fuzzy Expert Systems," *Fuzzy Expert Systems*, edited by A. Kandel, CRC Press, Boca Raton, FL, 1991.

[SCHN95] Schneider, N., "Dynamic Transfer of Control Between Manned and Unmanned Simulation Actors," Masters Thesis, AFIT/GE/ENG/95D-XX, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1995.

[SCHW91] Schwartz, D., G., "A System for Reasoning with Imprecise Linguistic Information," International Journal of Approximate Reasoning, Vol 8, 1991, 463-468.

[SHEA92] Sheasby, S., M., "Management of SIMNET and DIS Entities in Synthetic Environments," Masters Thesis, AFIT/GCS/ENG/92D-16, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1992.

[SHOE85] Shoemake, K., "Animating Rotation with Quaternion Curves," SIGGRAPH 85 19(3), San Francisco, CA., 245-254, 1985.

[SHOE93] Shoemake, K., "Quaternions," SIGGRAPH 93 Course Notes, #60, Anaheim, CA., J1 - J8. 1993.

[SOLT93] Soltz, B., B., " Graphical Tools for Situational Awareness Assistance for Large Battle Spaces. Interest," Masters Thesis, AFIT/GCS/ENG/93-21, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1993.

[STYT94] Stytz, M. R., and others, "Providing Situational Awareness Assistance to Users of Large-Scale, Dynamic, Complex Virtual Environments," *Presence*, 2(4), Fall 93, 297-313

[STYT95] Stytz, M., R., and E. Block, "Distributed Interactive Virtual Environments: Design, Implementation, and Experience," Air Force Institute of Technology, Wright-Patterson AFB, OH, 1995.

[SWIT92] Switzer, J. C., "A Synthetic Environment Flight Simulator: The AFIT Virtual Cockpit," Master's Thesis, AFIT/GCS/ENG/92D-17, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1992.

[TAMB95] Tambe, M. and others, "Intelligent Agents for Interactive Simulation Environment," AI Magazine, Spring 1995, 15-39.

[THOR88] Thorpe, J., "Warfighting with SIMNET - A Report From the Front," *Proceedings of the 10th Interservice/Industry Training Systems Conference*, Orlando, FL, 1988, 263 - 273.

[USAF92] "Basic Aerospace Doctrine of the United States Air Force," US Government Printing Office, 1992.

[VAND94] Vanderburg, J., C., "Space Modeler: An Expanded, Distributed Virtual Environment for Space Simulation," Masters Thesis, AFIT/GCE/ENG/94-XX, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1994.

[WILS93] Wilson, K., G., "Synthetic BattleBridge: Information Visualization and User Interface Design Applications in a Large Virtual Reality Environment," Masters Thesis, AFIT/GCS/ENG/93D-26, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1993.

[WINS92] Winston, P. H., *Artificial Intelligence, 3 ed*, Addison-Wesley, Reading, MA, 1992.

[ZADE65] Zadeh, L., A., "Fuzzy Sets," Information and Control, Vol 8:238-253,Academic Press, New York, June 1965.

[ZADE75] Zadeh, L., A., "The Concept of a Linguistic Variable and its Application to Approximate Reasoning," Information Sciences, Vol 8, 1975, 199-249 and 301-357.

[ZADE79] Zadeh, L., A., "A Theory of Approximate Reasoning," Machine Intelligence, edited by J. E. Hayes and others, Elsevier, 149-194.

[ZADE92] Zadeh, L., A., "Fuzzy Sets," in *Readings in Fuzzy Sets for Intelligent Systems*, edited by D. Dubois, H. Prade, and R. Yager, Morgan Kaufmann, San Mateo, CA, 1992.

[ZADE94] Zadeh, L., A., "Fuzzy Logic, Neural Networks, and Soft Computing," Communications of the ACM, 37(3), March 1994, 77 - 84.

[ZIMM87b] Zimmermann, H. -J., "Chapter 2 - Individual Decision Making in Fuzzy Environments," *Fuzzy Sets, Decision Making, and Expert Systems*, Kluwer Academic Publishers, Boston, MA, 1987, 1 - 14.

Vita

Capt Mark Edwards [redacted]. In 1982 he entered Clarkson University on a four-year AFROTC scholarship to study Electrical and Computer Engineering. Upon graduation and commissioning in 1986, Capt Edwards attended the basic Communications-Computer Systems Officer training course at Keesler AFB, MS where he was a Distinguished Graduate. His first permanent assignment was the Air Force Weapons Laboratory (now Phillips Laboratory) at Kirtland AFB, NM. While at Kirtland AFB, Capt Edwards earned a Masters of Science in Industrial Engineering from New Mexico State University. In 1991, Capt Edwards was transferred to Headquarters, Strategic Air Command, Offutt AFB, NE. Upon the deactivation of HQ SAC in 1992, Capt Edwards moved to the Air Force Global Weather Central, also at Offutt AFB. While at AFGWC, Capt Edwards attended Squadron Officers School at Maxwell AFB, AL and was a member of the Chief of Staff Trophy flight, the highest honor that the school can bestow. Capt Edwards came to AFIT in 1994. His next assignment is with the Atmospheric Prediction Branch of the Phillips Laboratory, Geophysics Directorate, Hanscom AFB, MA. Capt Edwards is married to the former Cynthia Bortolan of South Windsor, CT. He and his wife have two children, [redacted]

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Master's Thesis

4. TITLE AND SUBTITLE
The Automated Wingman: An Airborne Companion for Users of DIS Compatible Flight Simulators

5. FUNDING NUMBERS

6. AUTHOR(S)
Mark M. Edwards
Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/95D-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Command
ESC/AVM
20 Schilling Circle
Hanscom AFB, MA 01731-2816

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A major problem encountered by users of distributed virtual environments is the lack of simulators available to populate these environments. This problem is usually remedied by using computer generated entities. Unfortunately, these entities often lack adequate human behavior and are readily identifed as non-human. This violates the realism premise of distributed virtual reality and is a major problem, especially in training situations. This thesis addresses the problem by presenting a computer generated entity called the Automated Wingman. The Automated Wingman is a semi-automated computer generated aircraft simulator that operates under the control of a designated lead simulator. and integrates distributed virtual environments with intelligence. Access to distirbuted virtual environments is provided through the DIS protocol suite while human behavior is obtained through the use of a fuzzy expert system and a voice interface. The fuzzy expert system is designed around a hierarchy of knowledgebases. Each of these knowledgebases contains a set of fuzzy logic based linguistic variables that control the actions of the Automated Wingman. The voice interface allows the pilot of the lead simulator to direct the activity of the Automated Wingman. This thesis describes the design of the Automated Wingman and presents the current status of its implementation.

14. SUBJECT TERMS
Distributed Interactive Simulation, Computer Generated Forces, Protocol Data Unit, Semi-Automated Forces (SAFOR), Virtual Battlefield, Fuzzy Logic

15. NUMBER OF PAGES
114

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.