IN-08
75761
P. 199

NASA Contractor Report 189573

NASA TSRV ESSENTIAL FLIGHT CONTROL SYSTEM REQUIREMENTS VIA OBJECT ORIENTED ANALYSIS

K.S. Duffy and B.J. Hoza

BOEING COMMERCIAL AIRPLANE GROUP
THE BOEING COMPANY
Seattle, Washington

Contract NAS1-18027
March 1992

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

NASA Contractor Report 189573

NASA TSRV ESSENTIAL FLIGHT CONTROL SYSTEM REQUIREMENTS VIA OBJECT ORIENTED ANALYSIS

K.S. Duffy and B.J. Hoza

BOEING COMMERCIAL AIRPLANE GROUP
THE BOEING COMPANY
Seattle, Washington

Contract NAS1-18027
March 1992

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

1.0 SUMMARY

This report documents the work conducted under the NASA Advanced Transport Operating System task contract, Contract NAS1-18027, Task Assignment 15. It describes the development of a system requirements definition for the flight control system of a commercial-type research aircraft of Langley Research Center's Advanced Transport Operating System program, the Transport Systems Research Vehicle, also known as the NASA 515.

The objective of this work was to develop a definition of the Transport Systems Research Vehicle flight control system requirements able to facilitate the research and development of alternate, more advanced software and possibly hardware architectures for modern transport aircraft flight control systems.

The NASA 515 is a highly modified Boeing 737-100 aircraft designed specifically to investigate advanced navigation, guidance, control, and display concepts. In the experimental configuration, the aircraft is flown from a research flight deck (mounted in the aft cabin of the aircraft) whose entire flight control system is under software control. Therefore, alternate software implementations can readily be investigated.

For the purposes of this report, the Transport Systems Research Vehicle flight control system is defined to be the baseline software for the NASA 515 research flight deck, including all navigation, guidance, and control functions, and primary pilot displays. At the beginning of this study, the system was built around two digital Norden PDP-11/70 computers, termed Norden #1 and Norden #2. Since that time, the system was modified and the Norden computers were replaced with DEC MicroVax computers. For this report the system will be described for the Norden computer configuration. Norden #1 is the host to a Sperry microprocessor display system which provides the Primary Flight, Navigation, and Engine Display formats; hence it is also referred to as the Displays host computer. Norden #2 hosts the guidance, navigation, and control software; hence it is also referred to as the Flight Management/Flight Control host computer.

The scope of the requirements definition contained herein is limited to a portion of the Flight Management/Flight Control computer functionality. Included is a discussion of the tasks required to increase the scope of the requirements definition and recommendations for follow-on research.

2.0 INTRODUCTION

There is a strong requirement for a new generation of avionics systems with a more integrated hardware and software structure. Future cockpits will incorporate a wide range of enhancements. Heavy application of artificial intelligence techniques can be expected to encompass the entire spectrum of crew station technologies: from data fusion to optimized display resource management, to real-time onboard maintenance and fault reporting, and even to the optimization of pilot physiological needs. There is a significant challenge to integrate symbolic and numeric computation in a real-time environment as well as to effectively allocate functions between man and avionics. In addition, advances in the development of distributed and parallel processing systems necessitate the careful allocation of processing tasks among the system's computing resources in order to realize the increased system performance, reliability, and flexibility that these architectures offer. (References [1], [2], [3], [4].)

In order to successfully address the function specification and allocation problem, both the application requirements and the system architecture should be considered. Consequently, an effective systems engineering analysis is required to first identify the system requirements. A good requirements analysis addresses a system's *true* requirements, that is, those essential features or capabilities that a system must possess in order to fulfill its purpose, regardless of *how* the system is implemented. Development of the best architecture for the application starts with a good *essential requirements* specification.

The objective of this work was to analyze the NASA Transport Systems Research Vehicle (TSRV) flight control system (FCS) and develop a system specification that would offer high visibility of the essential system requirements in order to facilitate the future development of alternate, more advanced software and hardware architectures.

Neither the original (i.e., essential) TSRV FCS requirements nor the evolution of the baseline software are well documented, thus the "essence" of the system was backed out of the existing implementation. The available documentation consists only of a somewhat cryptic software description document and a listing of the source code. Consequently, the specification produced is more influenced by the particular design preferences than would be a "top down" analysis for the development of an all new system. Furthermore, because the analysis was of an existing system, the scope of the requirements were limited to only those features actually implemented in the code. However, a good systems analysis should specify the (essential) requirements in such a way that they may be easily modified and/or extended to include new features.

The systems analysis approach used was based on the Object Oriented Analysis (OOA) methodology developed by Boeing. In short, the Boeing OOA approach unifies traditional systems analysis methods of information modeling, process modeling (e.g., Yourdon-DeMarco Structured Analysis), and behavior modeling (e.g., Ward-Mellor Real-Time Extensions) into a single more powerful method which synergistically combines the strengths of all three methods.

A Computer Aided Software Engineering (CASE) tool is extremely useful for the OOA methodology. The Cadre Technologies Inc. **team***work* tool was used on this project as it indirectly supports the Boeing OOA methodology and was readily available at Boeing. Because **team***work* does not directly support the Boeing OOA approach, various conventions were developed to adapt **team***work* to the OOA methodology. Furthermore, for this report, the **team***work* data representing the TSRV FCS requirements analysis were reformatted into a single, unified, object-oriented specification.

This report documents a partial set of TSRV requirements and includes:

1. A brief description of the existing baseline system
2. A discussion of the systems analysis approach used, including an overview of OOA and a description of the typographical conventions manifest in the printed analysis
3. The requirements specification itself
4. Recommendations for extending the scope of the analysis and for potential follow-on research activities

3.0 SYMBOLS AND ABBREVIATIONS

This section contains symbols and abbreviations used throughout this report, including those used in the requirements specification of Section 6. However, please note that it does not include "object identifiers." These are defined in the object class descriptions found in 6.1 (see also 6.4, Index of Identifiers).

ADCEP	"Attribute / Derived attribute / Common domain / Event / Process"
AFCS	Automatic Flight Control System
AFD	Aft Flight Deck
AGCS	Advanced Guidance and Control System
ATOPS	Advanced Transport Operating System
BIU	Bus Interface Unit
CADC	Central Air Data Computer
CASE	Computer Aided Software Engineering
CRT	Cathode Ray Tube (display monitor)
CWS	Control Wheel Steering
DAS	Data Acquisition System
DATAC	Digital Autonomous Terminal Access Communication
DEC	Digital Equipment Company
DEU	Display Electronic Unit
DFD	Data Flow Diagram
DMA	Direct Memory Access
DME	Distance Measuring Equipment
DR11-A or B	a general purpose DEC bus (UNIBUS) interface card; a type of DMA device
DU	Display Unit
EIU	Effector Interface Unit
EIU/SIU	Effector Interface Unit/Sensor Interface Unit
ENG	Engine display format
ERD	Entity-Relationship Diagram
FAA	Federal Aviation Administration
FCS	Flight Control System
FFD	Forward Flight Deck
FM/FC	Flight Management/Flight Control
FPA	Flight Path Angle
GTA	Ground Track Angle
HSB	High Speed Bus
IAS	Indicated Airspeed *or* Instrument Approach System
ILS	Instrument Landing System
MCP	Mode Control Panel (same as MSP)
MLS	Microwave Landing System
MSP	Mode Select Panel (same as MCP)
NASA	National Aeronautics and Space Administration
NASA 515	FAA designation for NASA's TSRV airplane
NCDU	Navigation Control and Display Unit
NAV	Navigation display format
Norden #1	Host computer for TSRV AFD display system
Norden #2	Host computer for TSRV AFD FM/FC functions
OOA	Object Oriented Analysis
OCCM	Object Class Communication Model
OCRD	Object Class Relationship Diagram

OCRM	Object Class Relationship Model
PFD	Primary Flight Display format
SID	Standard Instrument Departure
SIU	Sensor Interface Unit
SS	SID/STAR
STAR	Standard Terminal Arrival
STD	State Transition Diagram
TSRV	Transport System Research Vehicle
WX	Weather

4.0 SYSTEM DESCRIPTION

The system of interest is the flight control system (FCS) for the research flight deck of NASA's Transport System Research Vehicle (TSRV), a highly modified Boeing 737-100 commercial jet airplane, also referred to as the "NASA 515".

Hardware and software descriptions are included below only to provide a general overview of the system. Other references should be consulted for a more detailed understanding.

4.1 TRANSPORT SYSTEM RESEARCH VEHICLE (TSRV)

The TSRV is used as part of NASA's Advanced Transport Operating System (ATOPS) program established to investigate advanced navigation, guidance, control, and display concepts. In the full-up test configuration it is flown from a second flight deck mounted in the cabin of the aircraft and referred to in this report as the Aft Flight Deck (AFD). The AFD contains the basic controls required to fly the airplane and uses the advanced controls and displays in a completely realistic environment. The AFD pilots have essentially complete control of the airplane, although the AFD control authorities are limited and the forward flight deck (FFD) crew can take over control at any time. (References [5], [6].)

4.2 AFT FLIGHT DECK (AFD) ARCHITECTURAL FEATURES

The following description of AFD architectural features is based on References [5] and [7].

The AFD features programmable electronic Primary Flight Displays, Navigation Displays, and Engine Displays, a Navigation Control and Display Unit (NCDU), a glare-shield mounted Advanced Guidance and Control System (AGCS) Mode Select Panel (MSP), and Panel Mounted Controllers (PMCs) that take the place of conventional column and wheel controls. (Recently the PMCs were replaced with side arm controllers.) Also, a Data Acquisition System (DAS) provides real-time data recording and data display for inflight and postflight evaluation.

Figure 4.1 is a simplified block diagram showing the arrangement of the principle components of the research system. The system is built around two Norden digital flight computers (militarized versions of the general purpose PDP-11/70 computer), designated Norden #1 and Norden #2. Both computers are interfaced to the AFD and an extensive array of sensors by a Digital Autonomous Terminal Access Communication (DATAC) system, a high-speed multitransmitter/receiver data bus.

Norden #2, the Flight Management/Flight Controls (FM/FC) host computer, receives sensor information and switch/button settings from the Sensor Interface Unit (SIU) across the DATAC bus. Based on the modes selected on the MSP, and the flight path selected via the NCDU, it performs the flight management, guidance, navigation, and flight controls functions which generate

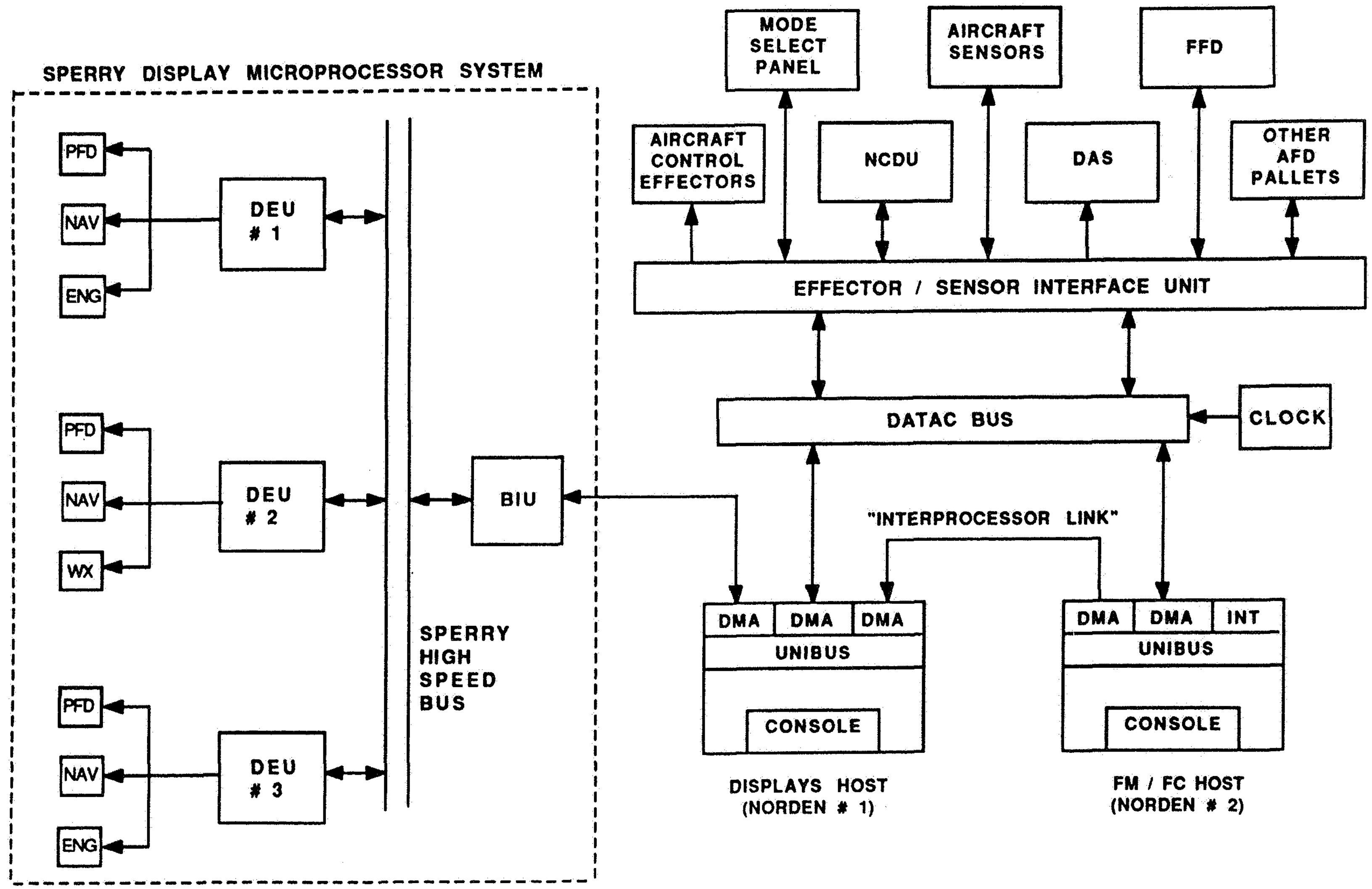


Figure 4.1 TSRV Aft Flight Deck Hardware Architecture

commands that are issued to the control surfaces via the Effector Interface Unit (EIU) to direct the airplane in flight. In addition, the active and provisional guidance buffers as well as other data needed to generate displays are shipped to Norden #1 via a DR11-B interprocessor link.

Norden #1, the Displays host computer, interprets input information from Norden #2 as well as from the sensors to create data blocks that are sent to a microprocessor-based Sperry Color Display System which provides the Primary Flight, Navigation, and Engine Display formats.

The Sperry system communicates with the displays host through a Direct Memory Access (DMA) channel. The information from the host is processed by Intel 80186 microprocessors which generate data usable by Sperry display units (DUs) for the creation and positioning of display symbology. There are twelve individual pieces of hardware involved in the standard Sperry display configuration. Included are eight color display units (denoted PFD, NAV, or ENG in Figure 4.1), three display electronic units (DEUs), and one bus interface unit (BIU). Each DEU has four microprocessors containing programs which perform I/O functions and create data in formats acceptable to the DUs. Note that one of the microprocessors contained in DEU #2 is a weather (WX) radar processor which feeds weather radar to each DEU.

4.3 BASELINE SOFTWARE

For the purposes of this report, the TSRV FCS is defined by the "baseline" software for the TSRV research flight deck including all navigation, guidance, and control functions, and primary pilot displays. It is comprised of the software contained in the Displays host computer (Norden #1), the FM/FC host computer (Norden #2), and the Sperry Color Display System.

The baseline software available for analysis on this project was a 1989 source code listing provided by NASA on a magnetic tape. This source code is best described by Reference [7], a software description document, also provided by NASA. The software described in Reference [7] is delivery 4.1 (D4.1) of the baseline FM/FC system which was released in July 1987.

It should be noted that these resources represent various portions of the software current on different dates. Consequently, some inconsistencies were found during the analysis. In these instances, the most logical and/or expedient resolution of the conflict was made; no attempt was made to document these conflicts or their resolution.

5.0 APPROACH

5.1 SYSTEMS ANALYSIS

This report addresses only part of the development of a complex, integrated system: the analysis and specification of its requirements. The role of "systems analysis" in the life-cycle of a system may be best understood in the proper context. Figure 5.1 shows one view of the evolution of a system, from the perception of a problem or objective to a fully operational solution.

Initially, a problem generally exists as ideas about what the issues are as well as even preconceived notions of the solution; or at best, as a partial list of requirements. It is necessary to move from this nebulous condition to a more precise set of requirements, or better yet, a well-defined *model* of the requirements. This model should be a complete, consistent, and coherent specification of the *essential* requirements of the system. *Analysis* is the process which identifies the "essence" of a system. See Reference [8].

The concept of essence helps distinguish between the true requirements of a system and those things which are really implementation oriented and/or technology dependent. In the analysis stage, the specification of the system should not be influenced by past design solutions -- "We've always done it that way" -- or by current technology constraints such as processor speeds or memory limitations. It is important to clearly define *what* the system needs to do, not *how* it shall do it.

Following the analysis phase is the design phase, where real-world constraints as well as design preferences are imposed upon the system's essential requirements specification. The result is a technology dependent model of a solution to the problem. It represents one answer to the question, "How can the essence of this system be realized?"

In the final phase of development, the design requirements are implemented in the form of hardware, software, etc.

In the case of the TSRV FCS, the systems analysis process started from a specification of the system "as built". Therefore, in a "reverse engineering" sense, the requirements were backed out of the existing implementation. Nevertheless, the goal was the same: to produce a specification of the essential system requirements.

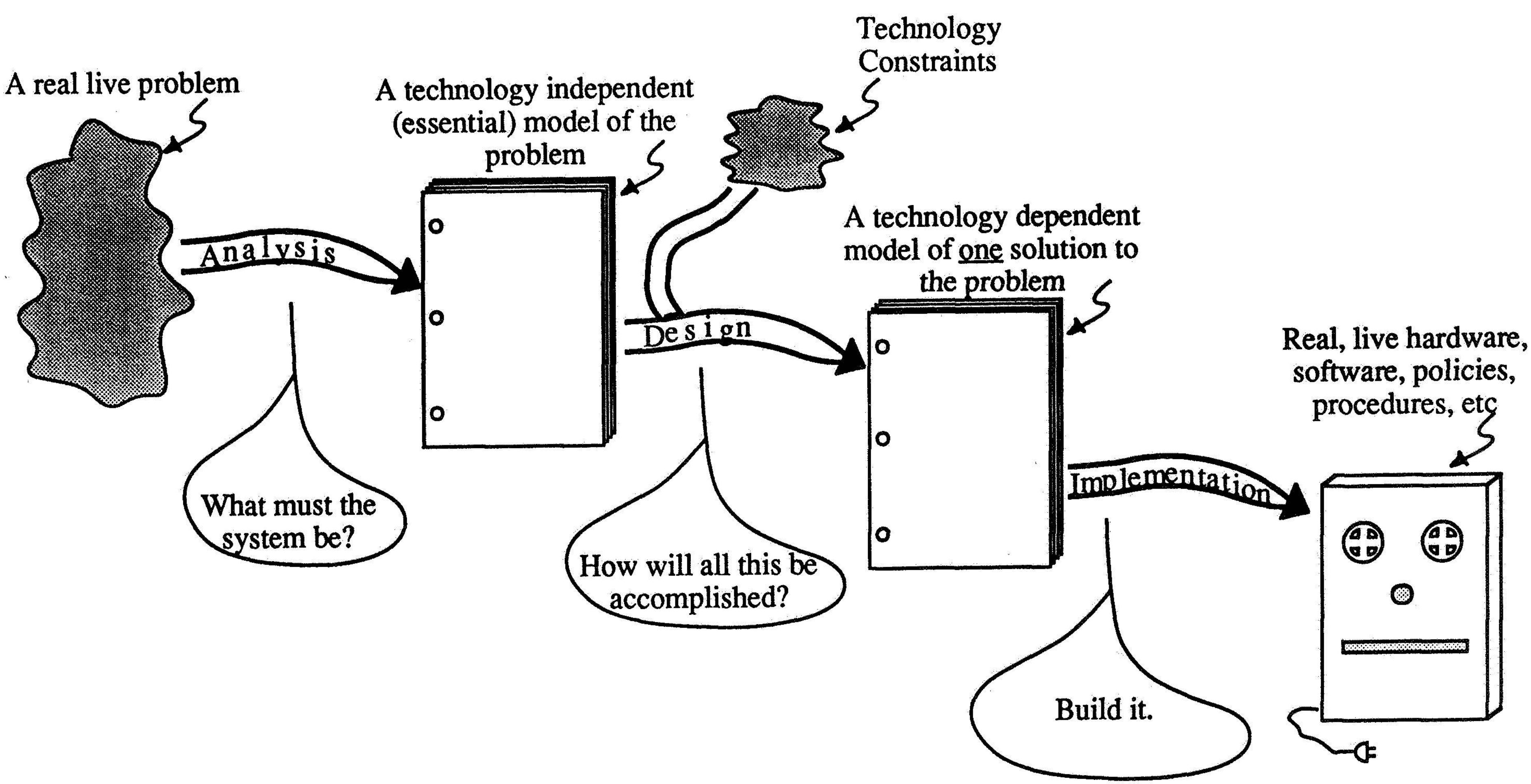


Figure 5.1 Systems Analysis in Context

5.2 METHODOLOGY SELECTION

The analysis of the TSRV FCS was carried out using a Boeing-developed Object Oriented Analysis (OOA) methodology. See Reference [9]. The Boeing OOA method unifies traditional systems analysis methods of information modeling, process modeling (e.g., Yourdon-DeMarco Structured Analysis), and behavior modeling (e.g., Ward-Mellor Real-Time Extensions) into a single more powerful method which synergistically combines the strengths of all three methods. See References [8], [10], [11], and [12] for background information.

OOA is an alternative to "structured analysis" as a requirements specification language. It unifies elements of information, behavior, and processing on an atomic level (object level), whereas structured analysis provides three hierarchical system level views which are not necessarily consistent and require parallel maintenance. OOA provides a method to rigorously define the requirements of any complex system in such a way that the analyst can readily identify any missing or contradictory requirements. The result is a complete, consistent, and simplified description of a system's essential requirements.

Furthermore, OOA provides a solid foundation for partitioning systems into "objects" and results in a system definition which allows a smooth transition into design. Object oriented specifications map very easily into object-oriented design and object-oriented programming (using languages such as Ada, Objective C, and Smalltalk), but also may be readily translated into other design schemes such as a standard hierarchical structured design.

OOA's partitioning methodology and design flexibility are a few of the features which make it particularly suited to the goal of defining alternate architectures for an existing complex system.

5.3 OBJECT ORIENTED ANALYSIS (OOA) OVERVIEW

OOA partitions a system into *objects*, or *object classes*. The goals of OOA are 1) to assist in the "discovery" of the object classes in a system and the relationships between those object classes; and 2) to provide a mechanism to document those object classes so that they can be verified and later communicated to designers and programmers. The "products" of OOA and the relationships between them are depicted in Figure 5.2. The products are the deliverable results of applying OOA to a specific system. In other words, they are the graphic and textual models that are produced by the process of analysis.

The products of OOA describe the system at two conceptual levels of detail: *class level* models and *system level* models. Class level models provide the detailed internal views of single object classes. System level models, on the other hand, describe the way object classes fit together to make the system as a whole. The system level models are not concerned with the internal details of the object classes. Likewise, the internal "schematic" of each object class must not concern itself with any other object class directly (i.e., knowledge of the internal workings of another object class is not allowed). These rules support the principle of information hiding.

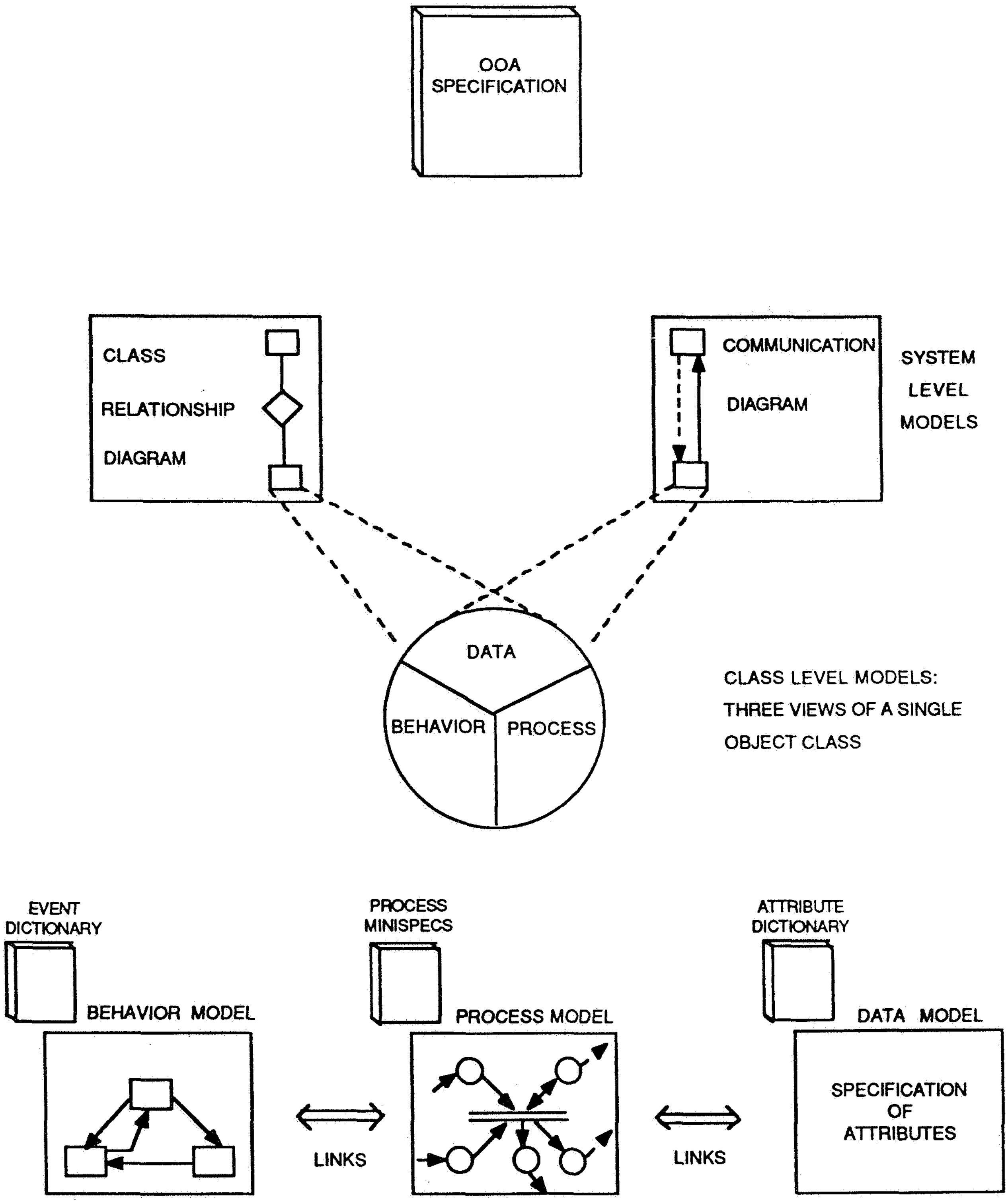


Figure 5.2 The Products of OOA

5.3.1 Class Level Models

An object class can be thought of as a template that represents the generic form of all objects which belong to that class. Object classes can be described by the data they carry, the processes they perform, and the behavior they exhibit. Therefore, each object class template is comprised of three class level models: 1) a *Data Model*; 2) a *Process Model*; and 3) a *Behavior Model*. In order for an object to be a member of a given class, it must conform to the class' template: that is, it must be described by the prescribed data, it must do the prescribed work (processing), and it must exhibit the prescribed behavior.

Data Models

The data aspect of an object class is described textually by a set of *Attributes*. Each attribute represents a single meaningful fact that is pertinent to, and necessary for, all instances of that class. The set of attributes for an object class defines the information that is relevant for that class.

Process Models

The Process Model captures the functionality of an object class, that is, the activities or work performed by the objects in that class. The model defines the set of *Processes* that operate upon or manipulate objects within that class.

A Process Model is described both graphically and textually. The graphical portion is in the form of the familiar dataflow diagram notation. A rigorous specification of the work performed by the processes is also necessary. The Structured Analysis "minispec" is a convenient mechanism for describing the details of a process.

Behavior Models

The behavior aspect of an object class is described by a finite state automaton. This can be thought of as a collection of *States* and *Transitions* which respond to certain *Events*. A state is a phase, stage, mode, or condition of being. The states for an object class are distinguished by ongoing processing, recognized events, and the responses to those events. An event is a circumstance, occurrence, or happening which may cause a change in the state of an object. A transition is a change from one state to another state due to a particular event.

5.3.2 System Level Models

There are two system level models that show how the population of object classes in the system are interconnected. The Object Class Relationship Model (OCRM) and the Object Class Communication Model (OCCM) depict how the object classes interrelate and interact, respectively.

The OCRM is represented by the Object Class Relationship Diagram (OCRD) (which resembles the Entity Relationship Diagram of traditional Information

Modeling methods). This model describes the existence relationships among object classes. A rough example of a relationship is, "every time there is an airport (object) there will be one or more runway (objects) associated with it."

On the other hand, the OCCM depicts the dynamic communication which takes place between objects during the life of the system. A rough example is: "the human pilot (object) may -- at some time -- send the following commands (in the form of data defined in the appropriate Data Model) to the autopilot (object): ....". In other words, this model shows the sources and destinations of every event and dataflow that can pass between objects. In the case where the number of communications that take place in a system render a single communication diagram of the whole system unreadable, multiple communication diagrams can be developed, each focusing on a related subset of the communications in the system.

5.3.3 External Object Classes

It should be noted that a complete understanding of the internals (data, behavior, processing) of *every* object class is generally not required, and sometimes may not be possible. This gives rise to a special type of object class, an *external* object class. All that need be known about them is that they are the sources and/or destinations of named events and data for the system.

The external object classes are identified in the analysis because they are significant in the overall understanding of the system. Later, these classes may be represented in the design phase by I/O interface routines. In the analysis phase, every object class must either be described by data, process, and behavior models, or be designated as an external object class.

5.4 ANALYSIS STRATEGY

A strategy serves the purpose of guiding the analyst toward the goal of completing the analysis, and different strategies may be appropriate in different situations. The analysis strategy used here was necessarily oriented toward the OOA methodology. The strategy was to develop the OOA models (see 5.3) of the system requirements using an information centered approach.

With this strategy, the analyst focuses first on the information aspect of a system in order to discover as many of the object classes as possible. The result of this activity is a first approximation of the OCRM along with Data Models for each of the object classes found so far.

Note that at this stage, those objects with little or no data (i.e., those which are more process and/or behavior intensive) have not necessarily been discovered: the list and definitions of the object classes are likely to be incomplete at this juncture. As subsequent phases of analysis (e.g., behavior and process modeling) are performed, new object classes may be discovered, or a few existing ones may be combined or split up. This is to be expected given a "bottom up" strategy of discovering object classes; the existence and completeness of an object class cannot truly and finally be validated until its Data, Process and Behavior Models are all defined.

Next, the analyst models the behavior aspect of the system by developing a Behavior Model for each object class. Additional behavior requirements not expressed by the Behavior Models for the known object classes may hint at the existence of hidden (i.e., as yet undiscovered) object classes that need to be exposed or at the need to re-partition some of the existing object classes.

The next step involves modeling the processing aspect of the system by developing a Process Model for each object class. The approach is to examine every state and every transition defined in the Behavior Model for every object class and ask the question, "Is anything supposed to happen in this state or on this transition?" Processes are then created to represent the work required and those processes are coupled with the appropriate states or transitions. Again, additional processing requirements may expose new object classes.

Finally, the OCCM of the system is built. In this strategy, the development of the communication model is a relatively mechanical process as no new information should be discovered. The task should be a simple matter of connecting the outputs of each object class to the appropriate inputs of the others.

The description of this strategy implies a relatively step-by-step approach. In reality, it was a more iterative process. Nevertheless, it was a useful strategy, particularly for an analysis of an existing functional implementation where the existence and partitioning of objects was not obvious.

5.5 CASE Tool

A Computer Aided Software Engineering (CASE) tool is extremely useful for the OOA methodology. A CASE tool can be thought of as a database management application which includes graphics objects and freeform text entities in its database schema, includes editors for the manipulation of these, and provides appropriate navigation throughout the database. The Cadre Technologies Inc. **team***work* family of tools (Reference [13]) was used as it was readily available at Boeing.

The **team***work* CASE tool family automates standard structured methodologies using interactive computer graphics and multi-user workstation power. Using **team***work* the analyst can examine the system being developed from multiple viewpoints -- all within one integrated environment that monitors the entire project's progress, rigorously checks for errors and produces automatic documentation. **Team***work*'s checking facilities provide automatic verification of the models by checking for completeness of the models, checking for consistency between models, and helping to find syntax errors.

Team*work* analysis tools help create descriptions and structured analysis of complex specifications. They allow description in both functional and object-oriented analysis. However, **team***work* does not directly support an object oriented analysis approach. Consequently, various conventions were developed to adapt **team***work* to the OOA methodology. Creative utilization of **team***work* has allowed essential information not directly supported by **team***work* to be recorded in the requirements specification (Section 6). Details regarding these conventions are described in 5.6.

Also, for the purposes of this report, a Boeing-developed **team***work* database report writer was used to format the TSRV FCS requirements analysis into a single, unified, object-oriented specification (Section 6).

5.6 CONVENTIONS

When reviewing the requirements specification given in Section 6, it may be helpful to know beforehand some details regarding the conventions that have been followed. This sub-section documents the conventions present in the printed reports and diagrams that comprise the requirements specification. It does not attempt to explain all of the rationale behind various conventions, but rather is intended to be a concise resource for the reader of the specification. For this reason, notation that has been published and is generally known is not explained. Furthermore, familiarity with the concepts of OOA is assumed (see 5.3), while knowledge of **team***work* (see 5.5) is not.

Team*work* did not directly support the Boeing OOA methodology. Although special application of the standard notation and conventions were defined by the tool developer to support Project Technology's Real-Time Object Oriented Analysis (see Reference [12]), these conventions were lacking in various respects. For example, one naming convention that required object class names to contain parentheses and spaces made it difficult to navigate within the software and eliminated any chance of producing an error report. Consequently, various conventions were developed to adapt **team***work* to the Boeing OOA methodology. Reference [14] contains additional background on some of the basic problems as well as solutions associated with using off-the-shelf CASE tools for Object Oriented Analysis.

Any convention typically satisfies a number of demands simultaneously, though perhaps not all of them ideally. A compromise is often required among content, readability, ease of use, and **team***work*'s "happiness" with the idea. The conventions outlined below maximize content but avoid sabotaging the automation offered by **team***work* (such as error reports, listings, on-line navigation among elements in the specification). At the same time, this supplementary-to-**team***work* information is captured in a rigorous fashion, amenable to mechanical (i.e., automated) syntax checking and cross-referencing. Some of the conventions are conventions required by **team***work*. Others are Boeing-developed, and have enabled the recording of essential information in an OOA specification that is not directly supported -- and therefore not expected -- by **team***work*.

The following paragraphs are designed to allow the reader of the specification to extract important information at a glance.

PLEASE NOTE:

> THE EXAMPLES PRESENTED HAVE BEEN TAKEN FROM THE REQUIREMENTS SPECIFICATION. HOWEVER, SOME HAVE BEEN MODIFIED TO BETTER ILLUSTRATE VARIOUS POINTS.

5.6.1 General Conventions for All Names

Some examples of various name conventions are:

an object name:	"FP--Flight_Plan"
an object name:	"WPT--Waypoint"
an object name:	"WFP--Waypoint_ona_Flight_Plan"
an attribute name:	"wfp-1-Ordinal_Position"
an attribute name:	"wfp-1-Flight_Plan_Name-R01A"
an attribute name:	"wpt-1-Latitude-id02"
a derived attribute name:	"fp-2-Horizontal_Guidance_Possible"
an event name:	"tpg-3-Mode_Reversion"
a process name:	"Maintain_Selected_Altitude"
a relationship id:	"R05"
a relationship name:	"R05A-may_be_assigned_to"

Words are generally capitalized and separated by an underscore ('_'); exceptions are allowed for small words such as "is", "or", etc. Relationship names are also an exception.

Names always begin with a letter and contain only letters, numbers, underscores and dashes.

The dash ('-') is used only when explicitly required by these conventions; it is not used otherwise.

A given name is generally made up of two or more *parts*, separated by the reserved dash character. One of these parts will be the *basic name* of the thing in question; that is, the 'common' word or phrase which is used in the analysis to uniquely identify the thing. The other parts, preceding and/or following this basic name, will embody useful or even necessary information. **Team***work*, however, considers the name of the thing to be the full string of characters, and is thus unaffected by the augmentation. The full string of characters is sometimes referred to as the *official name*, or as simply the *name* of the object.

Object names always begin with capital letters followed by a double dash. Attribute, derived attribute names and event names always begin with lower case letters, followed by "-1-", "-2-", and "-3-", respectively. Relationship ids are always of the form 'Rnn'. Relationship names always begin with the form 'RnnA' or 'RnnB' followed by a single dash.

5.6.2 Conventions for Object Classes

Every object class has a *name*, a *description*, and the following models: a *Data Model* and/or a *Behavior Model* and/or a *Process Model*. An object class' *official name* contains within it various flags and indicators. Its *description* is of a generic member of the class. The *description* is precise enough to enable a person to determine whether a candidate real-world object is -- or is not -- a member of the object class. It is a brief, imprecise English-language summary of the schematic template provided by the Data, Behavior and Process Models.

Several conventions were followed regarding the representation of object class templates in the specification. These include some which relate to the building of a name for a class, the format of its description, *primary identifier*, and *aliases*, and its appearance on system level diagrams. These whole-object conventions are explained in detail in 5.6.2. Conventions for the various parts of an object class template (i.e., those associated with the parts of the class level models) are documented in 5.6.3, 5.6.4, and 5.6.5.

Conventions for Object Class Names

Refer also to 5.6.1, General Conventions for All Names.

Some examples of *object class names* are:

"FP--Flight_Plan"
"WPT--Waypoint"
"WFP--Waypoint_ona_Flight_Plan"

The official object name consists of two parts: the *object id* and the *basic name*; these are separated by a pair of dashes. (A double dash is required to ensure that the object name immediately precedes its attributes' names in the case-insensitive alphabetic listing of **team***work* names.)

The *object id* is always present, always unique, generally mnemonic, and always entered in all capital letters. It is typically made from the first letters of the object's name ("FP" for "Flight Plan", "WFP" for "Waypoint on a Flight Plan"), though not always ("WPT" for "Waypoint"). This id is always two or more characters, and always begins with a letter.

The *basic name* is always in the singular, naming a typical instance of the object class.

Conventional Format for Object Class Descriptions

> *(regular object class)* A real-world Navigational Aid (aka Nav Aid) is a radio broadcast device located on the surface of the earth designed to enable an aircraft to determine its latitude and longitude (so long as the aircraft is within the service range of the Nav Aid). The NAVAID object class represents the knowledge within the system that real Navigational Aids exist in the real world; members of the NAVAID object class represent knowledge within the system of particular real-world nav aids. In other words, this is NOT an external object class: the system does NOT communicate (directly) with physical, real-world, on-the-ground nav aids. (Rather, the system communicates only directly with aircraft on-board receivers and other sensors.)
>
> The NAVAID object class is a subtype of the ATCWPT object class.
>
> The value of the navaid-1-Name is the official name designated by the FAA.

The description is always present, indented from the left margin.

The precise *kind* of object class being defined is given in italics within parentheses immediately prior to the body of the description. The following entries for object class kind may appear: regular object class; anonymous object class; external object class; external, anonymous object class; anonymous object class, no attributes; external, anonymous object class, no attributes.

The description itself is free-form text of one or more paragraphs.

Conventional Format for Object Class Primary Identifiers

> Primary Identifier: navaid-1-Name

Any non-anonymous object class must have one or more identifiers.

The chosen *primary* identifier is introduced with the key phrase "primary identifier". When several attributes form the primary id, each attribute is separated by a plus sign ("+"), and should appear on a line of its own; the last attribute may be followed by a period ("."). Comment blocks, if present, are delineated with a begin and end asterisk ("*").

Non-primary identifiers are indicated via attribute naming conventions. Refer to 5.6.3.

Conventional Format for Object Class Aliases

Alias:	Nav Aid, navaid;

An object class may have aliases. If so, these are indicated using the format shown above.

The Appearance of Object Classes on the OCRD

Please note that the attribute list which appears inside the rectangle of a given object class on the Object Class Relationship Diagram (OCRD) is enclosed in a comment block -- it has been entered manually, and **team***work* has not checked for agreement between this list and the list which is part of the object's definition. It may help to know this in case a subsequent change is required, or an inconsistency should arise.

The Appearance of Object Classes on OCCDs

Please note that just the *object id* appears within the object class rectangles on Object Class Communication Diagrams (OCCDs), to conserve diagram space. Also note that, by convention, a pair of asterisks following such an object id indicates that the context of the object class is not fully defined on that particular diagram; that is, there may be some data and/or events not shown on the diagram which this object class consumes or provides.

5.6.3 Conventions for Attributes, Derived Attributes, Common Domains, Events, and Processes

Several conventions were followed regarding the representation of attributes, derived attributes, events and processes in the analysis. These include some which relate to the building of a name for one, the format of its definition, and its appearance on diagrams. The notion of a *common domain* also aided in concise representation.

Every attribute has a *name* and a *definition*. An attribute's *official name* contains within it various flags and indicators (see **Conventions for ADCEP Names**). Its *definition* captures two things: the *domain* of values which the attribute can take on; and the precise *meaning* of those values for this attribute. The former is captured via a *domain specification*; the latter via an English *description* (see **Conventional Format for ADCEP Definitions**).

Common domains are created whenever two or more attributes share a common domain of values, due to an *essential* similarity (not a coincidence). This is done to follow the guideline, "Document each fact in just one place." In these cases, the group of similar attributes simply *reference* the common domain definition by its name, rather than repeat the shared *domain specification*. Like attributes, every common domain has a *name* and a *definition*.

Attributes which *reference* other definitions are said to have *nonprimitive definitions*. Attributes come by nonprimitive definitions in three ways: by referencing *common domains* as described above, by being *referential attributes*, or by being *compound attributes*. An attribute which directly specifies a domain of values (i.e., which contains its own *domain specification*) is said to have a *primitive definition*.

A *common domain* usually has a *primitive definition* (i.e., usually contains its own domain specification). However, some common domains are *compound domains*, and these have *nonprimitive definitions*. Compound domains reference other common domains (never attributes).

Derived attributes are identical to (regular) attributes except in an information modeling (data normalization) sense: they represent not stored data, but data derived from a process. (The process which derives a particular derived attribute may be explicitly defined for clarity or use in the Behavior Model, but is often left as an implied, always-available process. The term *dataflow* is not inappropriate.) Like attributes and common domains, every derived attribute has a *name* and a *definition* which consists of a *description* and a *domain specification* (which may be *primitive*, *nonprimitive*, or *compound*).

Every *event* has a *name* and a *description*. The description of an event captures the precise meaning of the event when it occurs. (There is no domain specification, since an event just occurs in a moment in time and is then gone.)

Every *process* has a *name* and a *description* (also known as a *minispec*). The description of a process captures *what* the process must accomplish, not *how* it will be accomplished: it is not algorithmic. (There is no domain specification since it makes no sense to speak of the values of the process itself.)

Attributes, derived attributes, common domains, events and processes are quite similar in regards to the conventions followed for them. It is therefore useful to refer to "*ADCEPs*". An *ADCEP* is "an attribute / derived attribute / common domain / event / process", i.e., either an attribute or a derived attribute or a common domain or an event or a process.

Conventions for ADCEP Names

Refer also to 5.6.1, General Conventions for All Names.

Some examples of *ADCEP names* are:

an attribute name:	"fp-1-Name"
an attribute name:	"wpt-1-Location_Description"
an attribute name:	"wpt-1-Longitude-id02"
an attribute name:	"wpt-1-Latitude-id02"
an attribute name:	"wfp-1-Flight_Plan_Name-R01A"
a derived attribute name:	"fp-2-Horizontal_Guidance_Possible"
an event name:	"tpg-3-Mode_Reversion"
a process name:	"Maintain_Selected_Altitude"
a common domain name:	"domn-Airspeed"

The official ADCEP name consists of at least two parts: first (except for process names), the *object id* of the object class which is the source of the ADCEP in question (or, in the case of a *common domain*, the characters "domn" stand in the place of the *object id*); next, the *basic name* of the ADCEP; next, zero or more *referential attribute tags*; finally, zero or more *identifier tags*. All parts are separated by a single dash.

It is a consequence of the above naming convention that a global dictionary printout (including both object classes and ADCEPs), alphabetized without regard to case, will list an object class's attributes, derived attributes and events immediately after listing the object class. Also, all *common domains* will be defined in a contiguous block. This considerably improves the readability and usefulness of such a printout, and aids the analysis process.

The *object id* (and "domn") is always entered in all lower case letters. This simply improves readability by markedly distinguishing an ADCEP name from an object class name (entered in all capital letters).

The *basic name* of the ADCEP signifies the meaning of the values in the domain of the ADCEP. (For events, it signifies the meaning of the event. For processes, it signifies the work accomplished by the process.) The meaning may be more precisely defined within the ADCEP's definition.

Referential attributes will include one or more *referential attribute tags*. These tags will be of the form 'RnnA' or 'RnnB', and serve to (a) flag an attribute as referential; and (b) indicate which relationship(s) are referenced, that is, from which related object class the attribute was borrowed. (Refer to 5.6.6 for conventions for relationships.)

For example, the reader immediately knows that the attribute "wfp-1-Flight_-Plan_Name-R01A" is a referential attribute, because it has the 'R' tag "R01A". Specifically, the reader sees that relationship "R01" is referenced; and that the object class connected to line "R01A" is the object class from which the attribute comes. ("FP--Flight_Plan" in this case.) Therefore, the reader knows that the attribute "wfp-1-Flight_Plan_Name-R01A" is in an essential sense equivalent to one of the attributes from the object class "FP--Flight_Plan". Exactly which attribute has been borrowed from the object class cannot be discerned from the attribute name itself; it is documented via the attribute's *definition* (see **Conventional Format for ADCEP Definitions**).

Attributes which form part of an identifier (a.k.a. key) will include one or more *identifier tags*. These tags name the identifier (primary, secondary, tertiary, ...) of which an attribute is a part. The tag will be of the form 'idnn'. For example, the attribute "wpt-1-Latitude-id02" forms part of a secondary identifier for the "WPT--Waypoint" object class. Note, however, that *primary* identifiers are not always indicated in this way, because the analyst is required by **team***work* to indicate a chosen primary identifier using a different method (see **Conventional Format for Object Class Primary Identifiers** in 5.6.2).

Conventional Format for ADCEP Definitions

An ADCEP definition captures two things: the domain of values which the ADCEP can take on (the *domain specification)*; and the meaning of those values for this ADCEP. (Events and processes have no domain specification.) While the meaning is often clear from the name of the ADCEP, a more precise description is usually helpful. The domain of values is described in *essential* (technology independent) terms; how these values might be stored in memory or on disk is of no concern in analysis.

An ADCEP definition consists of a well-structured body of text. The six examples below illustrate some of the various kinds of definitions. They are followed by the details of the exact structure.

1. A nonprimitive definition of an attribute of an object class:

airspdhmc-1-Selected_Airspeed
(attribute)

This is the pilot-indicated airspeed which shall be maintained while the Airspeed_Hold_Mode_Controller is engaged.

domn-Airspeed.

2. A primitive definition of an attribute of an object class:

rw-1-Usable_Length
(attribute)

The usable length of the Runway, starting from the Threshold.

Type:	Numeric;
Range:	from 250 to 30,000 feet;
Accuracy:	to 1 foot;

3. A primitive definition of a common domain:

domn-Airspeed
(common domain)

Type:	Numeric;
Range:	from 45 to 750 feet per second;
Accuracy:	to 3 significant digits;

4. A nonprimitive definition of a common domain:

domn-Latitude
(*compound common domain*)

	domn-Latitude_Degrees
+	domn-Latitude_Hemisphere.

5. An event definition:

hpg-3-Engagement_Criteria_Satisfied
(*broadcast event*)

This event corresponds to a determination that the HPG can successfully capture the horizontal component of the flight plan.

6. A process definition:

Capture_and_Hold_Selected_Altitude
(*process*)

This process generates a vertical acceleration steering command so as to smoothly change the actual altitude of the aircraft to match the pilot-selected altitude.

Every definition begins with the ADCEP name in bold print on a line by itself. The precise *kind* of ADCEP being defined is given in italics within parentheses on the next line, indented 0.125 inches. The following may appear:

- attribute; referential attribute; subtype indication; compound attribute; compound referential attribute;
- derived attribute; compound derived attribute;
- broadcast event; directed event;
- process;
- common domain; compound common domain.

An ADCEP definition usually has a description, indented 0.5 inches from the ADCEP name, and consisting of free-form English text. Except for events and processes, the description captures the precise meaning of the information recorded by the ADCEP. This description is commonly a brief, incomplete sentence like those of a normal dictionary entry, which assumes the reader mentally forms the sentence, "A [name] is [description]." Another common format is to begin the description with "This..."; the word refers to the ADCEP name. In the case of an event, the description records the precise meaning of the event's occurrence. In the case of a process, the description records unambiguously *what* the process must accomplish without getting into *how* it might be implemented. The description for an ADCEP may be omitted where obvious from the name. The ADCEP's domain specification (if any) is enclosed in a rectangle indented from both margins about 1 inch. (Events and processes never have domain specifications; other ADCEPs must.)

There are two kinds of domain specifications: primitive and nonprimitive. The ADCEP itself is said to have either a primitive or nonprimitive definition, accordingly.

Nonprimitive ADCEP Definitions

A *nonprimitive definition* references other ADCEPs in its *domain specification.* Generally, just a single ADCEP is referenced; occasionally, two or more may be referenced.

Team*work* checks referenced ADCEPs when validating a *nonprimitive definition.*

When two or more ADCEPs are referenced, this indicates that the ADCEP is actually *composed of* two or more meaningful parts, each with its own domain.

domn-Latitude
(compound common domain)

	domn-Latitude_Degrees
+	domn-Latitude_Hemisphere.

Reference to this ADCEP is, strictly speaking, a shorthand for reference to its parts (the true, atomic ADCEPs). Such an ADCEP is called a *Compound ADCEP* -- specifically, a *compound attribute, compound derived attribute, compound domain,* and so on.

If a referenced ADCEP is an attribute of another object class (its name begins with an *object id*, not with "domn-"), as opposed to being a common domain, then the message is:

> *this attribute is identical in meaning and domain to the referenced attribute; it is, in essence, the same attribute.*

wfp-1-Waypoint_Name-R01B
(referential attribute)

The id of the Waypoint which is on the named Flight_Plan by means of this WFP.

wpt-1-Name.

Referential attributes are always defined in this way.

If a referenced ADCEP is a *common domain* (its name begins with "domn-"), then the message is:

> *this ADCEP has the same domain of values as the referenced* common domain, *but the precise meaning of those values for this ADCEP is documented in this definition's* description.

airspdhmc-1-Selected_Airspeed
(attribute)

This is the pilot-indicated airspeed which shall be maintained while the Airspeed_Hold_Mode_Controller is engaged.

domn-Airspeed.

domn-Airspeed
(common domain)

Type:	Numeric;
Range:	from 45 to 750 feet per second;
Accuracy:	to 3 significant digits;

Common domains are created whenever two or more ADCEPs share a common domain of values, due to an *essential* similarity (not a coincidence). This is done to follow the guideline, "Document each fact in just one place."

Primitive ADCEP Definitions

A *primitive definition* is one that does not reference other ADCEPs; its *domain* (if any) is explicitly stated via a *domain specification.* (Events and processes never have domain specifications; they are said to have primitive definitions.)

The *domain specification* is not checked by **team***work*, but does adhere to a rigorous syntax amenable to mechanical checking by auxiliary programs.

The format of the *domain specification* varies, depending on the kind of domain being described: enumeration, subtype indication, string, numeric, or integer. However, all formats begin by indicating which of these five domain specifications follows ("Type").

Enumerable domains are defined simply by listing the valid values.

navaid-1-Type
(attribute)

This indicates the class of the nav aid.

Type:	Enumeration;
Values:	VOR, VOR-DME, TACAN, NDB, Localizer_Beacon;

An attribute in a supertype which names the subtype object class is called a *subtype indication*, and its domain of valid values is, by definition, the object ids of the subtype object classes.

wfp-1-Method_of_Acquisition-R03
(subtype indication)

The manner in which the aircraft will move from the previous WFP to this WFP.

Type:	Subtype Indication;
Objects:	DA, DMEAA;

An ADCEP whose domain of valid values is strings of characters may be defined by a maximum size, minimum size, legal characters and/or format (if any).

navaid-1-Name
(*attribute*)

This is the official identifier of the nav aid, as assigned by the FAA.

Type:	String;
Range:	exactly 3 upper case letters;

wpt-1-Location_Description
(*attribute*)

A short English description of the location of the waypoint, typically the name of a nearby city.

Type:	String;

rw-1-Name
(*attribute*)

This is the official name of the Runway, which is determined by (1) its Magnetic_Heading, and (2) its position at the Airport relative to a runway parallel to it (if one exists). In the format "nh", 'n' is a string of 1 to 2 digits equal to the magnetic heading rounded to the nearest 10 degrees; 'h' is either "L" or "R" (for left-hand runway and right-hand runway, respectively). If there is no parallel runway, then the 'h' is dropped.

Type:	String;
Range:	from 1 to 3 characters;
Format:	nh;

The description of a string ADCEP may have to include an English description of its required format in order to be precise.

Numeric ADCEPs are described by range and accuracy, as in the following examples.

rw-1-Usable_Length
(*attribute*)

The usable length of the Runway, starting from the Threshold.

Type:	Numeric;
Range:	from 250 to 30,000 feet;
Accuracy:	to 1 foot;

domn-Magnetic_Variation
(*common domain*)

An amount of variation between actual and measured magnetic north; a correction factor to be applied to compass readings.

Type:	Numeric;
Range:	from -180 to +180 degrees;
Accuracy:	to 0.1 degrees;

domn-Airspeed
(*common domain*)

Type:	Numeric;
Range:	from 45 to 750 feet per second;
Accuracy:	to 3 significant digits

domn-Altitude_Above_Mean_Sea_Level
(*common domain*)

Type:	Numeric;
Range:	from 1500 to 45,000 feet;
Accuracy:	to 4 significant digits;

Integer ADCEPs are described by range only.

wfp-1-Ordinal_Position_on_Plan
(*attribute*)

The sequential position (first, second, third, ...) of this WFP on the named Flight_Plan.

Type:	Integer;
Range:	from 1;

Note, however, that an ADCEP recorded to the nearest whole unit (such as rw-1-Usable_Length) is *not* an integer ADCEP. It is still essentially *numeric*.

NOTE: A pair of angle brackets ("<>") is used to denote "deliberately omitted" material. Such material was not uncovered during the analysis process. (Refer to Section 7 for a discussion of relevant recommendations.)

domn-Magnetic_Variation
(*common domain*)

An amount of variation between actual and measured magnetic north; a correction factor to be applied to compass readings.

Type:	Numeric;
Range:	from -180 to +180 degrees;
Accuracy:	to <> degrees;

The Appearance of Attributes on the OCRD

The attributes of an object class are listed inside the object class rectangle on the OCRD, below a line of dashes. Note, however, that this list is enclosed in a comment block -- it has been entered manually, and **team***work* has not checked for agreement between this list and the list which is part of the object's definition. It may help to know this in case a subsequent change is required, or an inconsistency should arise.

5.6.4 Conventions for Behavior Model Diagrams

Several conventions were followed regarding the representation of behavior via State Transition Diagrams (STDs). These include conventions which relate to the construction of state and transition labels and are explained in detail below.

Conventional Format for State Labels

```
ARMED *
------------------------------- *
E. Display_State
E. Watch_for_Path_Divergence
E. Evaluate_Engagement_Criteria
4
```

The state name is given on line 1. Line 2 consists of a separator line to improve readability; note that this line is enclosed in a comment block which begins after the state name. Below this appear the enabled processes, in the usual "E." notation. An id number for the state appears in the lower left corner.

State names are in all capital letters. Words are separated with spaces rather than underscores ("_").

Conventions for Transitions

A transition is a change from one state to another state due to a particular event. The state from which a transition is drawn is referred to as the *source state*; the state to where it goes, the *next state*. The source state of a transition is said to "contain" that transition. Every transition has either an *event block* or a *guard* (or both), and may additionally have a *response* (ordered sequence of triggered processes and signalled events).

Transition labels have been augmented in several ways in order to simplify otherwise busy STDs. A standard STD (i.e., a standard *OOA* STD) modified according to these conventions will generally have fewer transition lines and/or fewer state boxes. Please note, however, that the *meaning* of both diagrams is the same (i.e., the behavior specified is identical): the purpose of these conventions is to increase readability only. The transformation from an augmented to a standard version of a given STD is mechanical.

```
afd-3-Altitude_Hold_Mode_Request /
---------------------------------
automc-2-Engaged .and.
.not. gs-2-Engaged
---------------------------------
T. Set_Selected_Altitude_to_Actual
```

The *event block*, when present, is always first. A slash ("/") and a separator line follow the event block whenever a *guard* and/or *response* follow. The *guard*, when present, is always preceded and followed by a separator line, and always appears before the *response* (if any). The *response*, when present, always appears last, below a separator line.

Conventions relating to transition *event blocks* and *guards* are described below. No special conventions were adopted for the *response* on a transition.

Conventions for Transition Event Blocks

An event block consists of a list of *event names* and/or *boolean expressions*. A mixture is allowed. Each item *must* be separated by the key word ".or.". A transition which has a *boolean expression* in its *event block* is referred to as an *evaluated transition*.

When more than one item is listed, the transition is taken to be a shorthand for a set of transitions (one for each item) which share the same source state, next state, guard and response. A step in the construction of the standard version of an STD is to replace such a transition with just that set.

The *event name* carries the usual meaning: when an object of the class is in the source state, and the named event occurs, then the transition will be taken (unless the transition's *guard* is false -- see below).

A *boolean expression* is a series of one or more *relational expressions* separated by ".or." and/or ".and.". Each *relational expression* compares an attribute or derived attribute of an object to that of another (or to a constant) via the usual operators:, "<", ">", "=", etc. Relational expressions may be negated using ".not.", and may consist simply of an attribute or derived attribute name, so long as its domain is boolean.

The *boolean expression* becoming *true* while the object is in the source state is treated as an event in the usual sense. Note well, however, the following subtlety.

Upon entry into a state which contains one or more *evaluated transitions*, each of the boolean expressions must be evaluated (in any order), and the associated transition taken *immediately* if the guard evaluates to *true*; none of the enabled processes associated with the state shall begin until *after* all of these boolean expressions have been evaluated as *false*. This stipulation is extremely important, especially in light of the infinite speed with which an essential (technology independent) process is said to execute.

It is possible to create non-deterministic state models using *evaluated transitions*; however, it is also possible to create non-deterministic state models using standard OOA state models in which states contain multiple processes that evaluate criteria and signal events (refer to the paragraph below). It is up to the analyst, as usual, to ensure the correctness of the state model, which may or may not mean ensuring that the model is deterministic.

A step in the construction of the standard version of an STD is to relegate the evaluation of a *boolean expression* in an *event block* to a specially-created process which signals one of two specially-created events (one event for "[boolean expression] true," another for "[boolean expression] false." A specially-created "temporary evaluation" state for each original state containing one or more *evaluated transitions* would enable all the original state's specially-created processes, and would contain two transitions: one going into the original state (for the "false" event), and one going into the *next state* of the original *evaluated transition*. (The response on the former being null; the response on the latter being the response of the original.) All transitions going into the original state would be redirected to the specially-created state.

When one of the events occurs, or when one of the *boolean expressions* evaluates to *true*, it is said that *an event in the* event block *has occurred*.

Conventions for Transition Guards

A guard consists of a single *boolean expression*. A transition containing a guard is referred to as a *guarded event transition*, and cannot be taken unless the guard evaluates to *true* at the time an event in the *event block* occurs. If the guard evaluates to *false*, then the transition is not taken: the event in the *event block* is ignored. It is permissable for a guard to appear on a transition which has no *event block*. However, in this case, the guard is treated as an *event block* consisting of single *boolean expression*. See **Conventions for Transition Event Blocks** for more on *boolean expressions*.

As with *evaluated transitions*, the evaluation of the *boolean expression* can be relegated to a specially-created process which signals one of two specially-created events. Any *guarded event transition* would be modified by removing its original *response* and changing its *next state* destination to a specially-created "temporary evaluation" state which would enable the specially-created process, and would contain two transitions: one going back into the *source state* of the original *guarded event transition*, with null *response* (for the "false" event), and one going ahead into the original *next state* of the original *guarded event transition*, with *response* equal to that of the original.

5.6.5 Conventions for Process Model Diagrams

Part of the specification of a process may be to generate particular events when certain conditions are met. In this case, the process is shown to produce the event by means of a dotted arrow notation. Other than this, no special conventions were adopted for Process Model diagrams.

5.6.6 Conventions for Relationships

Several conventions were followed regarding the representation of relationships in the analysis. These include some which relate to the building of a name for a relationship, its definition, and its appearance on diagrams.

Throughout the discussion of these conventions, please refer to Figure 5.3, which shows an annotated example from the OCRD.

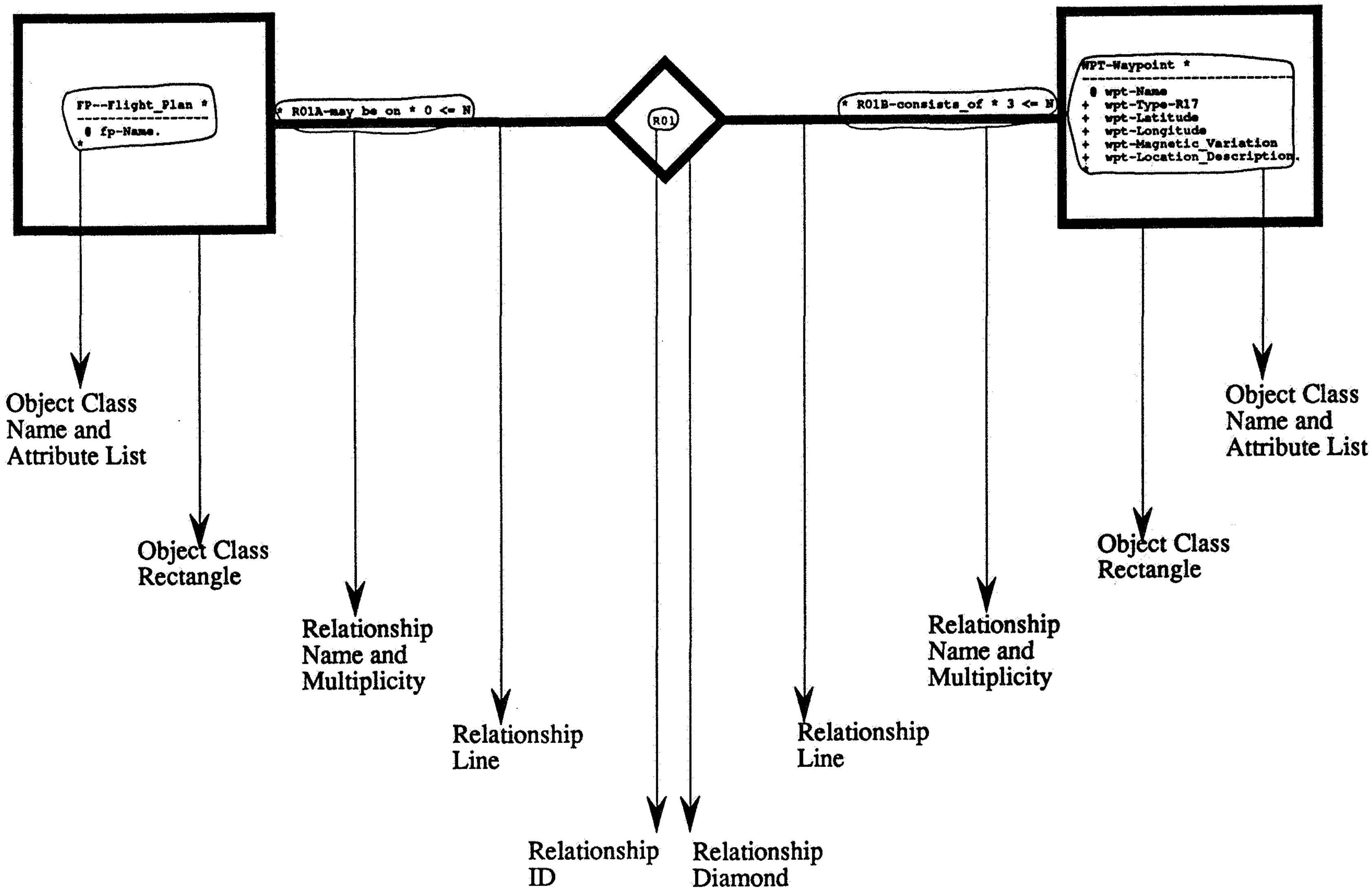


Figure 5.3 The Parts of a Relationship Defined

Every relationship has an *id*, two *names*, two *statements of multiplicity*, and a *definition*. A relationship's *id* is a simple identifier used by **team***work*. Each *name* is formed by prefixing a *relationship name id* to the *basic name* of the relationship itself. Each *name* (as well as each *statement of multiplicity*) is defined from the viewpoint of one of the object classes, so that one can refer to the *subject* and *object* of the name (or statement of multiplicity), in the grammatical sense of these words.

Each *basic name* signifies the meaning behind an association between members of the two object classes from the viewpoint of a generic member of the *subject* object class. Each *statement of multiplicity* captures the number of objects from the *object* object class that may be associated with a single object from the *subject* object class. The relationship's *definition* documents the precise details of a relationship, expanding on the meanings signified by the relationship as it appears on the Object Class Relationship Diagram (OCRD).

Conventions for Relationship Names

Refer also to 5.6.1, General Conventions for All Names.

Some examples of *relationship names* are:

"R01A-may_be_on"
"R01B-consists_of"
"R04A-centers_on"
"R04B-may_be_the_center_of"

The official relationship name consists of two parts: the *relationship name id* and the *basic name*. These are separated by a single dash.

The *relationship name id* is always present; it uniquely identifies one of the two *names* for the relationship. It is always formed by appending 'A' or 'B' to the *relationship id*. Each relationship is assigned a unique id of the form 'Rnn' (e.g., R01, R02, R23, R46), referred to as the *relationship id*. This id is the only way **team***work* will refer to the relationship. Keep this in mind when reading **team***work*-generated reports and listings. **Team***work* uses this id (and only this id) to alphabetize.

The *basic name* of the relationship is always a verb or verb phrase. It signifies the meaning of an association between members of the two object classes from the viewpoint of a generic member of the *subject* object class. The precise meaning attached to the association is documented in the relationship definition (see **Conventions for Relationship Definitions**). Which object class is the *subject* of the verb phrase, and which is the *object*, is obvious from the physical placement of the relationship name on the OCRD, but cannot be discerned from the name alone (see **The Appearance of Relationships on the OCRD**). These verb phrases might use the word "may" to underscore the conditionality of the relationship, though not always.

Conventions for Statements of Multiplicity

Some examples of *statements of multiplicity* are:

"0 <= N"
"3 <= N"
"0 <= N <= 2"

The *statement of multiplicity* follows a straight-forward, **team***work*-imposed syntax. It documents half of what is conventionally called the 'multiplicity of the relationship': the minimum and maximum number of objects in a class (the *object* object class) which might be associated, at any one time, with an object of the *subject* object class, defined from the viewpoint of a generic member of the *subject* object class. The other half of the 'multiplicity of the relationship' is documented in the other *statement of multiplicity*, in which the two object classes have reversed grammatical roles. For an example, refer to **Rendering an English Definition From OCRD Relationships.**

Conventions for Relationship Definitions

A textual plain-English relationship definition need not be given unless the definition will illuminate subtle aspects of reality or policy not obvious from the relationship as documented on the Object Class Relationship Diagram (OCRD).

The Appearance of Relationships on the OCRD

As shown in Figure 5.3, most aspects of a relationship are documented on the OCRD; only the *definition* is absent. (The definition serves to clarify the terse, somewhat artificial statements appearing on the OCRD.)

Recall that a relationship name is either a verb or a verb phrase. As shown in the figure, each name appears next to the object class which is the *object* of the verb phrase (in the grammatical sense), as is customary when using two names. For example, the relationship name which is derived from the viewpoint of a "WPT--Waypoint" is placed next to the object class rectangle for "FP--Flight_Plan"; the former object class is the *subject* object class, the latter the *object* object class.

The *statement of multiplicity* which appears next to each relationship name follows a self-evident, **team***work*-imposed syntax. The statement of multiplicity appears adjacent to the *object* object class, and is defined from the viewpoint of a generic member of the *subject* object class.

As shown, the *relationship name* and *statement of multiplicity* appear together, generally on a single line. The relationship name is enclosed in a comment block; **team***work* does not perform any checking of this name. However, **team***work* does inspect and expect statements of multiplicity.

The *relationship id* appears inside the relationship diamond. This id is the only way **team***work* will refer to the relationship. Keep this in mind when reading **team***work*-generated reports and listings. **Team***work* uses this id (and only this id) to alphabetize.

Rendering an English Definition From OCRD Relationships

It generally helps the reader of an OCRD gain an understanding of the meaning of a relationship by reading all parts together in two sentences, as follows.

1. Read the name of one of the object classes; an object class is always named in the singular.
2. Read the *relationship name* (verb phrase) which is next to the *other* object class.
3. Read the *statement of multiplicity* which follows the name, translating it into English.
4. Read the name of the other object class, treating it as the *object* of the verb (in the grammatical sense), switching to the plural form for one-to-many and many-to-many relationships.
5. Repeat this process from the other direction (starting with the other object class).

The relationship depicted in Figure 5.3 may thus be read as follows:

"A <u>Flight Plan</u> <u>consists of</u> <u>3 or more</u> <u>Waypoints</u>."
"A <u>Waypoint</u> <u>may be on</u> <u>zero or more</u> <u>Flight Plans</u>."

6.0 REQUIREMENTS SPECIFICATION

This section contains the actual requirements specification for the TSRV flight control system (see Section 4 for a system description). Although strictly speaking it is self-contained, any specification is best understood in the proper context. A key to a good understanding of the specification is an understanding of the typographical conventions used in it. Refer to 5.6. Of course, it is also imperative to understand the concepts of Object Oriented Analysis (OOA). Refer to 5.2 and 5.3

The scope of the requirements definition contained herein is limited to the Flight Management/Flight Control computer functionality. (Future analysis would surely expand this "system boundary" in order to fully specify the TSRV flight control system. Section 7 contains recommendations along these lines.) Within the scope of the system analyzed, some requirements were identified but not thoroughly analyzed. In this analysis, such requirements were captured by identifying special placeholder object classes.

Such an object class is termed a "pseudo external object class": treated as external until such time as this aspect of the system can be analyzed further. For the interim, this object class provides a source and sink for closely related events and data: those known to be required by objects already defined in the analysis. Further analysis may reveal that this single placeholder must become several object classes. Also, several of these pseudo external object classes might be split and joined.

The requirements specification is divided into four sub-sections. 6.1 specifies the object classes alphabetically by defining the class level models for each object class. 6.2 contains definitions for "Common Domains," which are referred to within object class specifications when defining the domain of valid values of particular data (see 5.6). 6.3 contains system level models: an Object Class Relationship Model and an Object Class Communication Model. Finally, 6.4 consists of an Index of Identifiers, which will aid in determining where an identifier is defined or referenced.

NOTE: it may be useful to consult the system level models to formulate a reading order of the class level specifications given alphabetically in 6.1.

6.1 CLASS LEVEL MODELS

This sub-section contains the complete specification of each object class, in alphabetical order. The requirements contained in the class level models are laid out in a format geared for maximum readability, although 5.3 and 5.6 may be useful in understanding the various products of OOA and the conventions used for the various pieces of a specification, respectively. The layout of these specifications and the explicit correspondence between the layout and the class level products of OOA are described below.

Each object class is defined within its own paragraph, 6.1.n (where 'n' is a number): its Data Model, Behavior Model, and Process Model are documented. The title to paragraph 6.1.n consists of the *official name* of the object class. Immediately following this title is the object class description. Following the description, are **Summary** and **Definitions**. If the object class has a Behavior and/or Process Model, the Behavior Model Diagram and/or Process Model Diagram appear after **Definitions**.

Summary consists of a summary of the object class: all attributes, derived attributes, events and processes are listed here without being defined. Also included is the *primary identifier* of the object class, along with any *aliases* by which the class is known. If the object class has a Behavior or Process Model, such is indicated.

Definitions contains all of the textual definitions relevant to the object class. It defines the attributes, derived attributes, events and processes (in this order). Thus the Data Model is completely contained in **Definitions** (and is summarized in **Summary**), while just the textual definitions associated with the Behavior Model and Process Model appear.

The Behavior Model Diagram and Process Model Diagram depict the Behavior Model and Process Model, in the forms of a State Transition Diagram (STD) and a Data Flow Diagram (DFD), respectively. These diagrams document essential information; they are not mere summaries of information present elsewhere in the specification.

However, every item referenced on these diagrams is defined textually in **Definitions** of the source object class specification. With respect to the STD, signalled events, triggered processes and enabled processes are listed in **Summary**, and defined in **Definitions**. Consumed events are defined in the source object class specification, as are attributes and derived attributes appearing in boolean expressions. The meaning of a state is taken to be self-evident from (and identical to) the context given in the STD; a textual description is not given. With respect to the DFD, signalled events, processes, and "outflow" attributes and derived attributes are listed in **Summary**, and defined in **Definitions**. Consumed ("inflow") attributes and derived attributes are defined in the source object class specification.

6.1.1 AAMC--AFD_AFCS_Mode_Controller

(*anonymous object class, no attributes*) The AAMC, when enabled, controls the highest modes of the Automatic Flight Control System (AFCS). It takes input from the AFD as well as from the various subordinate mode-controller objects (for automatic mode reversion).

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Event:	aamc-3-Disengage_Auto_Modes
Event:	aamc-3-Disengage_Default_Mode
Event:	aamc-3-Disengage_VCWS
Event:	aamc-3-Engage_Auto_Modes
Event:	aamc-3-Engage_Default_Mode
Event:	aamc-3-Engage_VCWS

Definitions

aamc-3-Disengage_Auto_Modes
(*directed event*)

This event corresponds to a command from the AAMC to the AUTOMC.

aamc-3-Disengage_Default_Mode
(*directed event*)

This event corresponds to a command from the AAMC to the DEFMC.

aamc-3-Disengage_VCWS
(*directed event*)

This event corresponds to a command from the AAMC to the VCWS.

aamc-3-Engage_Auto_Modes
(*directed event*)

This event corresponds to a command from the AAMC to the AUTOMC.

aamc-3-Engage_Default_Mode
(*directed event*)

This event corresponds to a command from the AAMC to the DEFMC.

aamc-3-Engage_VCWS
(*directed event*)

This event corresponds to a command from the AAMC to the VCWS.

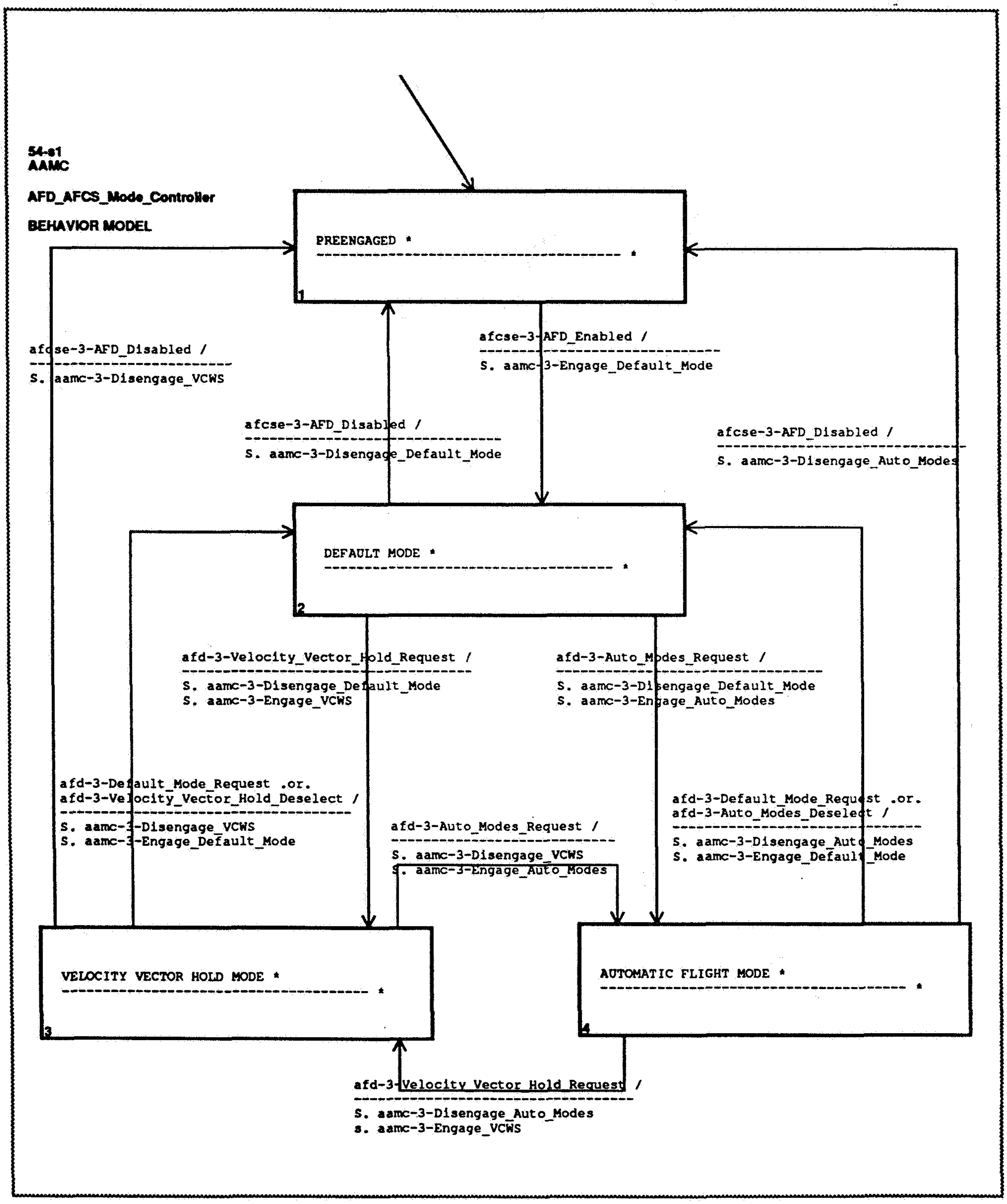


Figure 6.1 Behavior Model Diagram for AAMC

6.1.2 ACWS--Attitude_Hold_Control_Wheel_Steering_Mode_Controller

(*anonymous object class*) The Attitude_Hold_Control_Wheel_Steering_Mode_-Controller, while engaged, maintains the attitude of the aircraft by issuing commands to the Effector_Controller. The ACWS allows the AFD_Crew to change the attitude-being-maintained by responding to wheel/column inputs (or equivalent control inputs) while engaged.

Summary

Primary Identifier:	*none*
Alias:	ACWS Mode Controller
Behavior Model:	*present*
Process Model:	*present*
Attribute:	acws-1-Selected_Bank_Angle
Attribute:	acws-1-Selected_Pitch_Angle
Attribute:	acws-1-Selected_Yaw_Angle
Derived Attribute:	acws-2-Attitude_Hold_Engaged
Derived Attribute:	acws-2-Commanded_Bank_Angle
Derived Attribute:	acws-2-Commanded_Pitch_Angle
Derived Attribute:	acws-2-Commanded_Yaw_Angle
Process:	Set_Selected_Attitude_to_Actual
Process:	Maintain_Attitude_of_Aircraft

Definitions

acws-1-Selected_Bank_Angle
(*attribute*)

domn-Bank_Angle.

acws-1-Selected_Pitch_Angle
(*attribute*)

domn-Pitch_Angle.

acws-1-Selected_Yaw_Angle
(*attribute*)

domn-Yaw_Angle.

acws-2-Attitude_Hold_Engaged
(*derived attribute*)

"True" while the ACWS is engaged (i.e., controlling the aircraft so as to maintain an attitude); "false" when disengaged.

domn-Boolean.

acws-2-Commanded_Bank_Angle
(*derived attribute*)

domn-Bank_Angle.

acws-2-Commanded_Pitch_Angle
(derived attribute)

domn-Pitch_Angle.

acws-2-Commanded_Yaw_Angle
(derived attribute)

domn-Yaw_Angle.

Set_Selected_Attitude_to_Actual
(process)

This process makes the selected attitude (pitch, bank, yaw) of the aircraft equal to the instantaneous actual attitude.

Maintain_Attitude_of_Aircraft
(process)

This process commands the Effector_Controller to maintain the selected attitude (pitch, bank, yaw) of the aircraft, while at the same time allowing the Aft Flight Deck Crew to modify this selected attitude via control inputs.

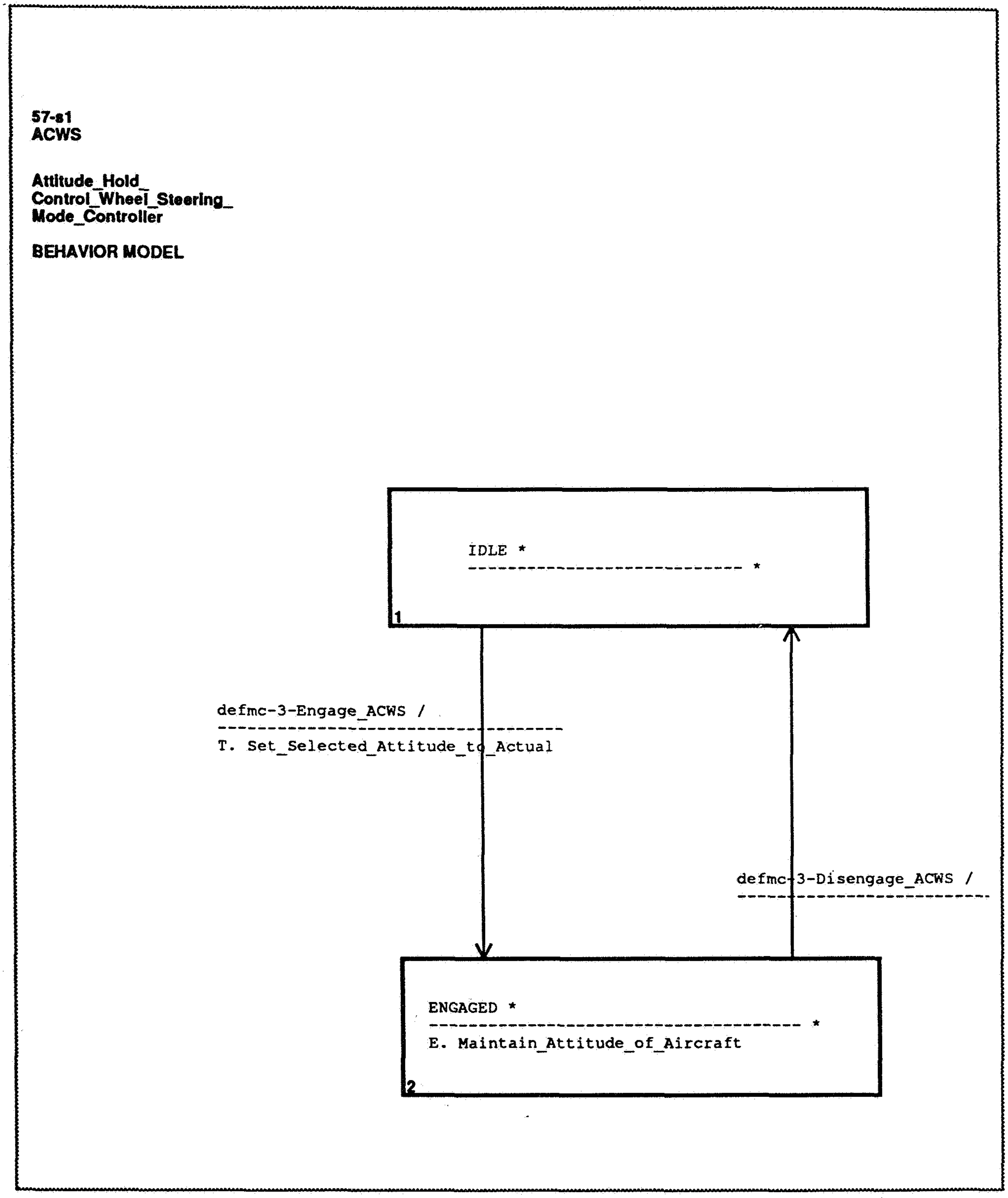


Figure 6.2 Behavior Model Diagram for ACWS

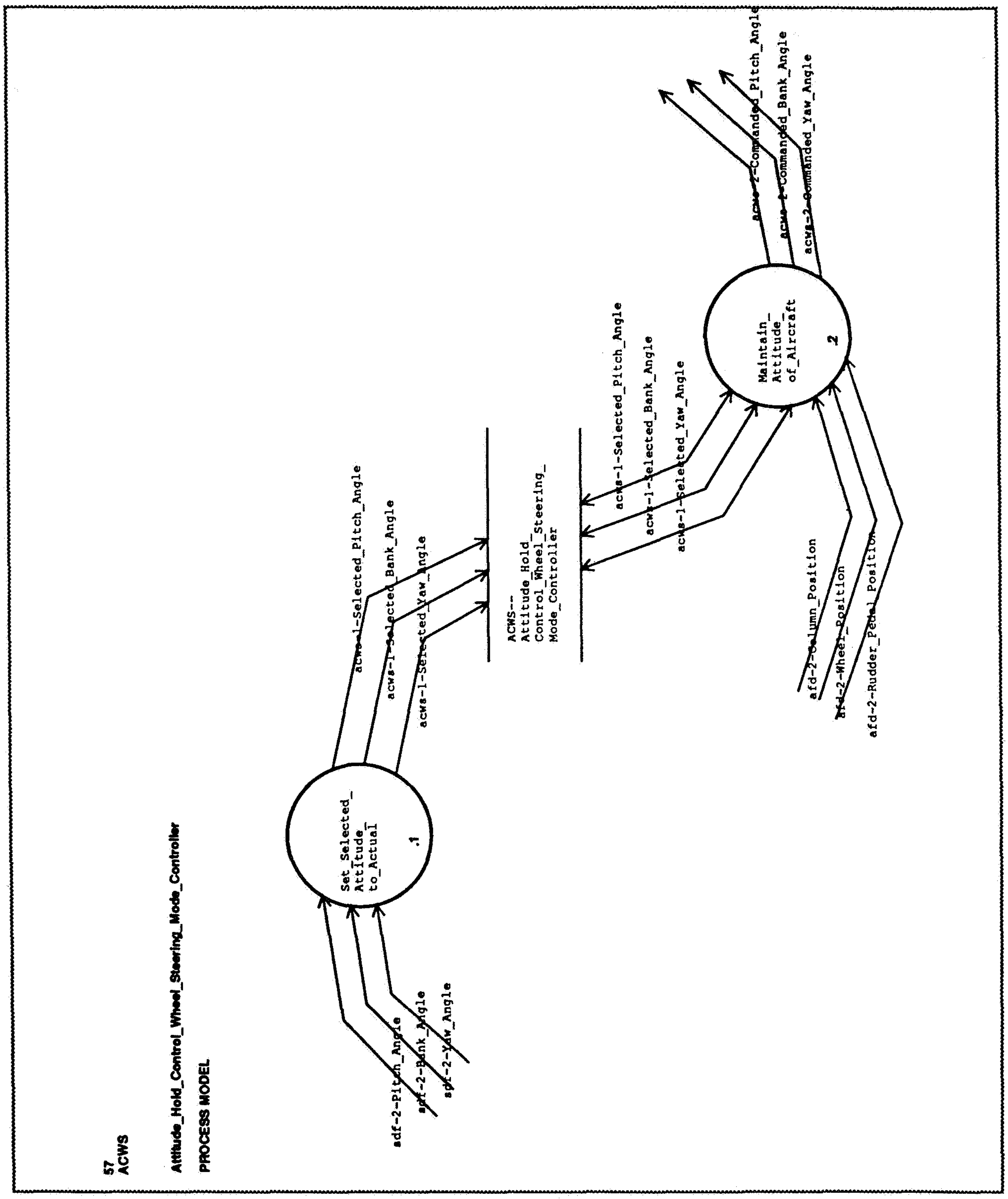


Figure 6.3 Process Model Diagram for ACWS

6.1.3 AFCSE--AFCS_Enabler

(anonymous object class, no attributes) The AFCSE enables and disables the Automatic Flight Control System (AFCS), taking input only from the Forward Flight Deck Crew.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Event:	afcse-3-AFD_Disabled
Event:	afcse-3-AFD_Enabled
Event:	afcse-3-FFD_Disabled
Event:	afcse-3-FFD_Enabled

Definitions

afcse-3-AFD_Disabled
(broadcast event)

This event is signaled at the moment the AFCSE disables the AFD.

afcse-3-AFD_Enabled
(broadcast event)

This event is signaled at the moment the AFCSE enables the AFD.

afcse-3-FFD_Disabled
(broadcast event)

This event is signaled at the moment the AFCSE disables the FFD.

afcse-3-FFD_Enabled
(broadcast event)

This event is signaled at the moment the AFCSE enables the FFD.

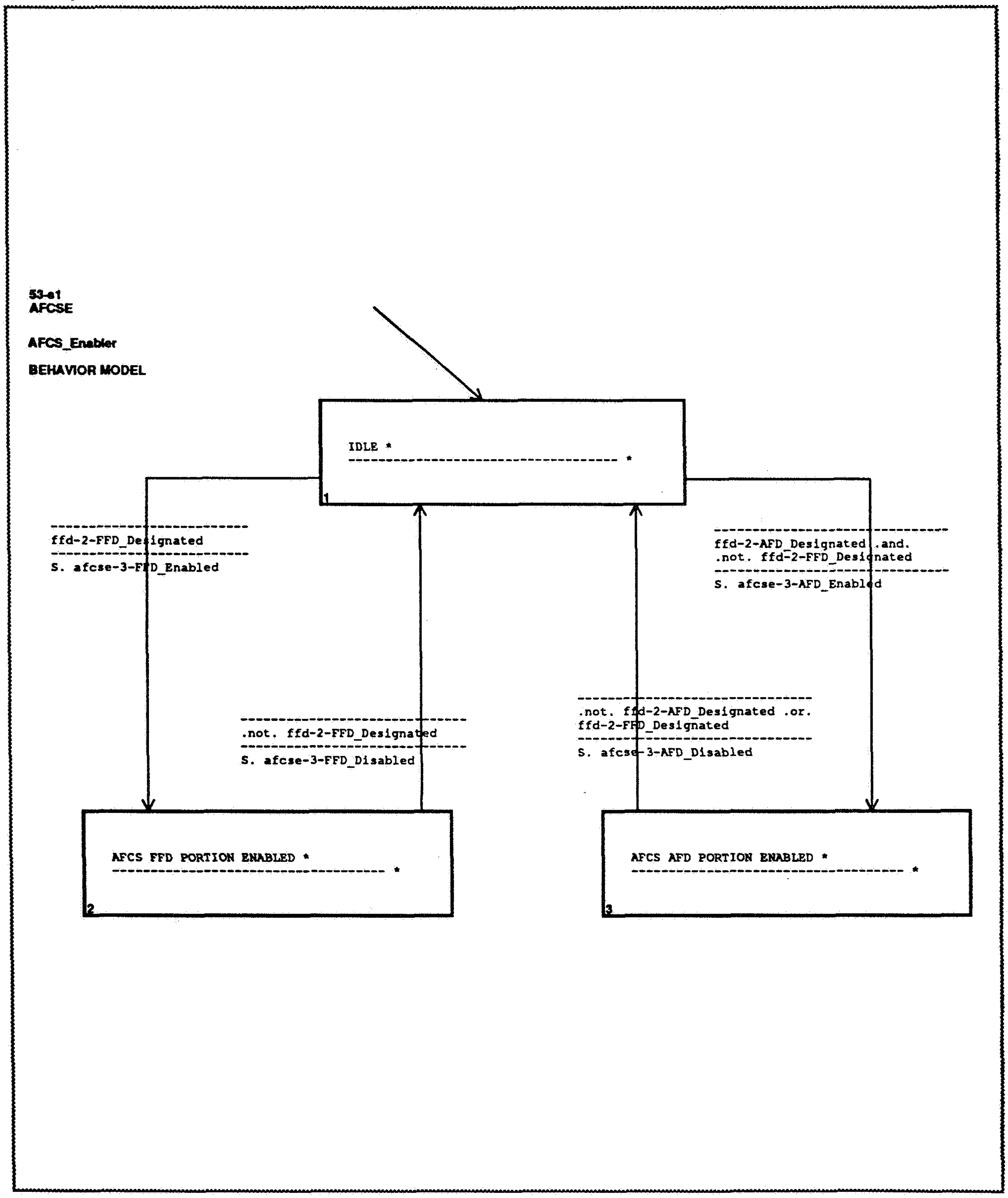


Figure 6.4 Behavior Model Diagram for AFCSE

6.1.4 AFD--Aft_Flight_Deck_Crew

(*external, anonymous object class, no attributes*) The Aft_Flight_Deck_Crew consists of the research pilots who fly the aircaft in the aft flight deck.

Summary

Primary Identifier:	*none*
Derived Attribute:	afd-2-Autothrottle_Engaged
Derived Attribute:	afd-2-Column_Position
Derived Attribute:	afd-2-Designated_Default_Mode
Derived Attribute:	afd-2-Rudder_Pedal_Position
Derived Attribute:	afd-2-Wheel_Position
Event:	afd-3-Airspeed_Decrement_Request
Event:	afd-3-Airspeed_Hold_Mode_Deselect
Event:	afd-3-Airspeed_Hold_Mode_Request
Event:	afd-3-Airspeed_Increment_Request
Event:	afd-3-Altitude_Decrement_Request
Event:	afd-3-Altitude_Hold_Mode_Deselect
Event:	afd-3-Altitude_Hold_Mode_Request
Event:	afd-3-Altitude_Increment_Request
Event:	afd-3-Attitude_Hold_Deselect
Event:	afd-3-Auto_Modes_Deselect
Event:	afd-3-Auto_Modes_Request
Event:	afd-3-Autothrottle_Deselect
Event:	afd-3-Default_Mode_Request
Event:	afd-3-Flight_Path_Angle_Decrement_Request
Event:	afd-3-Flight_Path_Angle_Hold_Mode_Deselect
Event:	afd-3-Flight_Path_Angle_Hold_Mode_Request
Event:	afd-3-Flight_Path_Angle_Increment_Request
Event:	afd-3-Ground_Track_Angle_Decrement_Request
Event:	afd-3-Ground_Track_Angle_Hold_Mode_Deselect
Event:	afd-3-Ground_Track_Angle_Hold_Mode_Request
Event:	afd-3-Ground_Track_Angle_Increment_Request
Event:	afd-3-Horizontal_Guidance_Deselect
Event:	afd-3-Horizontal_Guidance_Request
Event:	afd-3-Time_Guidance_Deselect
Event:	afd-3-Time_Guidance_Request
Event:	afd-3-Velocity_Vector_Hold_Deselect
Event:	afd-3-Velocity_Vector_Hold_Request
Event:	afd-3-Vertical_Guidance_Deselect
Event:	afd-3-Vertical_Guidance_Request

Definitions

afd-2-Autothrottle_Engaged

(*derived attribute*)

"True" when the AFD has engaged the autothrottle. Note: this derived attribute has been placed in the AFD object class until such time as the system can be analyzed further.

domn-Boolean.

afd-2-Column_Position
(*derived attribute*)

The control input which indicates the desired Pitch Angle of the aircraft.

domn-Column_Position.

afd-2-Designated_Default_Mode
(*derived attribute*)

The mode to which the AFCS should default when a higher mode reverts, or when the AFD selects "Default Mode"; the abbreviations stand for "Attitude Hold Control Wheel Steering" and "Manual Electric", resp.

Type:	Enumeration;
Values:	ACWS, ManEl;

afd-2-Rudder_Pedal_Position
(*derived attribute*)

The control input which indicates the desired Yaw Angle of the aircraft.

domn-Rudder_Pedal_Position.

afd-2-Wheel_Position
(*derived attribute*)

The control input which indicates the desired Bank Angle of the aircraft.

domn-Wheel_Position.

afd-3-Airspeed_Decrement_Request
(*broadcast event*)

This event corresponds to a request to decrease the maintained or preselected airspeed.

afd-3-Airspeed_Hold_Mode_Deselect
(*broadcast event*)

This event corresponds to a request to release control of the aircraft's airspeed; i.e., a request to disengage the AIRSPDHMC.

afd-3-Airspeed_Hold_Mode_Request
(*broadcast event*)

This event corresponds to a request to hold the current or preselected airspeed; i.e., a request to engage the AIRSPDHMC.

afd-3-Airspeed_Increment_Request
(*broadcast event*)

This event corresponds to a request to increase the maintained or preselected airspeed.

afd-3-Altitude_Decrement_Request

(broadcast event)

This event corresponds to a request to decrease the maintained or preselected altitude.

afd-3-Altitude_Hold_Mode_Deselect

(broadcast event)

This event corresponds to a request to release control of the aircraft's altitude; i.e., a request to disengage the ALTHMC.

afd-3-Altitude_Hold_Mode_Request

(broadcast event)

This event corresponds to a request to hold the current or preselected altitude; i.e., a request to engage the ALTHMC.

afd-3-Altitude_Increment_Request

(broadcast event)

This event corresponds to a request to increase the maintained or preselected altitude.

afd-3-Attitude_Hold_Deselect

(broadcast event)

This event corresponds to a request to disengage the "Attitude Hold with Control Wheel Steering (CWS)" mode; i.e., a request to disengage the ACWS Mode Controller.

afd-3-Auto_Modes_Deselect

(broadcast event)

This event corresponds to a request to disable the automatic flight modes (the "Auto" Modes).

afd-3-Auto_Modes_Request

(broadcast event)

This event corresponds to a request to enable the automatic flight modes (the "Auto" Modes).

afd-3-Autothrottle_Deselect

(broadcast event)

This event corresponds to a request to release automatic control of the throttle.

afd-3-Default_Mode_Request
(broadcast event)

This event corresponds to a request to engage the default mode of autopilot control, which might be the ACWS mode or the Manual Electric mode.

afd-3-Flight_Path_Angle_Decrement_Request
(broadcast event)

This event corresponds to a request to decrease the maintained or preselected flight path angle.

afd-3-Flight_Path_Angle_Hold_Mode_Deselect
(broadcast event)

This event corresponds to a request to disengage the FPAHMC.

afd-3-Flight_Path_Angle_Hold_Mode_Request
(broadcast event)

This event corresponds to a request to hold the current or preselected flight path angle.

afd-3-Flight_Path_Angle_Increment_Request
(broadcast event)

This event corresponds to a request to increase the maintained or preselected flight path angle.

afd-3-Ground_Track_Angle_Decrement_Request
(broadcast event)

This event corresponds to a request to decrease the maintained or preselected ground track angle.

afd-3-Ground_Track_Angle_Hold_Mode_Deselect
(broadcast event)

This event corresponds to a request to release control of the aircraft's ground track angle; i.e., a request to disengage the GTAHMC.

afd-3-Ground_Track_Angle_Hold_Mode_Request
(broadcast event)

This event corresponds to a request to hold the current or preselected ground track angle; i.e., a request to engage the GTAHMC.

afd-3-Ground_Track_Angle_Increment_Request
(broadcast event)

This event corresponds to a request to increase the maintained or preselected ground track angle.

afd-3-Horizontal_Guidance_Deselect

(broadcast event)

This event corresponds to a request to disengage the "Horizontal Path Guidance" mode; i.e., a request to disengage the Horizontal Path Guide.

afd-3-Horizontal_Guidance_Request

(broadcast event)

This event corresponds to a request to engage the "Horizontal Path Guidance" mode; i.e., a request to engage the Horizontal Path Guide.

afd-3-Time_Guidance_Deselect

(broadcast event)

This event corresponds to a request to disengage the "Time Path Guidance" mode; i.e., a request to disengage the Time Path Guide.

afd-3-Time_Guidance_Request

(broadcast event)

This event corresponds to a request to engage the "Time Path Guidance" mode; i.e., a request to engage the Time Path Guide.

afd-3-Velocity_Vector_Hold_Deselect

(broadcast event)

This event corresponds to a request to disengage the "Velecity Vector Hold with Control Wheel Steering (CWS)" mode; i.e., a request to disengage the VCWS Mode Controller.

afd-3-Velocity_Vector_Hold_Request

(broadcast event)

This event corresponds to a request to engage the "Velocity Vector Hold with Control Wheel Steering (CWS)" mode; i.e., a request to engage the VCWS Mode Controller.

afd-3-Vertical_Guidance_Deselect

(broadcast event)

This event corresponds to a request to disengage the "Vertical Path Guidance" mode; i.e., a request to disengage the Vertical Path Guide.

afd-3-Vertical_Guidance_Request

(broadcast event)

This event corresponds to a request to engage the "Vertical Path Guidance" mode; i.e., a request to engage the Vertical Path Guide.

6.1.5 AIL--Aileron

(external, anonymous object class, no attributes) The Aileron is one of the aerodynamic control effectors used to control the motion of the aircraft.

Summary

Primary Identifier: *none*
Derived Attribute: ail-2-Current_Position

Definitions

ail-2-Current_Position
(derived attribute)

This is the current sensed position of the aileron.

domn-Aileron_Position.

6.1.6 AIRSPDHMC--Airspeed_Hold_Mode_Controller

(*anonymous object class*) The Airspeed_Hold_Mode_Controller, when engaged, commands the Effector_Controller to capture and maintain a pilot-selected airspeed.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Attribute:	airspdhmc-1-Selected_Airspeed
Derived Attribute:	airspdhmc-2-Commanded_Airspeed
Process:	Display_Actual_Airspeed
Process:	Display_Selected_Airspeed
Process:	Set_Selected_Airspeed_to_Actual
Process:	Increment_Selected_Airspeed
Process:	Decrement_Selected_Airspeed
Process:	Display_State
Process:	Capture_and_Hold_Selected_Airspeed

Definitions

airspdhmc-1-Selected_Airspeed
(*attribute*)

This is the pilot-indicated airspeed which shall be maintained while the Airspeed_Hold_Mode_Controller is engaged.

domn-Airspeed.

airspdhmc-2-Commanded_Airspeed
(*derived attribute*)

This is the commanded airspeed which shall be attained.

domn-Airspeed.

Display_Actual_Airspeed
(*process*)

This process displays the Actual_Airspeed of the aircraft.

Display_Selected_Airspeed
(*process*)

This process displays the Selected_Airspeed.

Set_Selected_Airspeed_to_Actual
(*process*)

This process makes the Selected_Airspeed attribute equal to the instantaneous Actual_Airspeed.

Increment_Selected_Airspeed

(*process*)

This process increases the current Selected_Airspeed by 5 knots.

Decrement_Selected_Airspeed

(*process*)

This process decreases the current Selected_Airspeed by 5 knots.

Display_State

(*process*)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

Capture_and_Hold_Selected_Airspeed

(*process*)

This process commands the Surface_Controller to change the Actual Airspeed of the aircraft to match the Selected Airspeed.

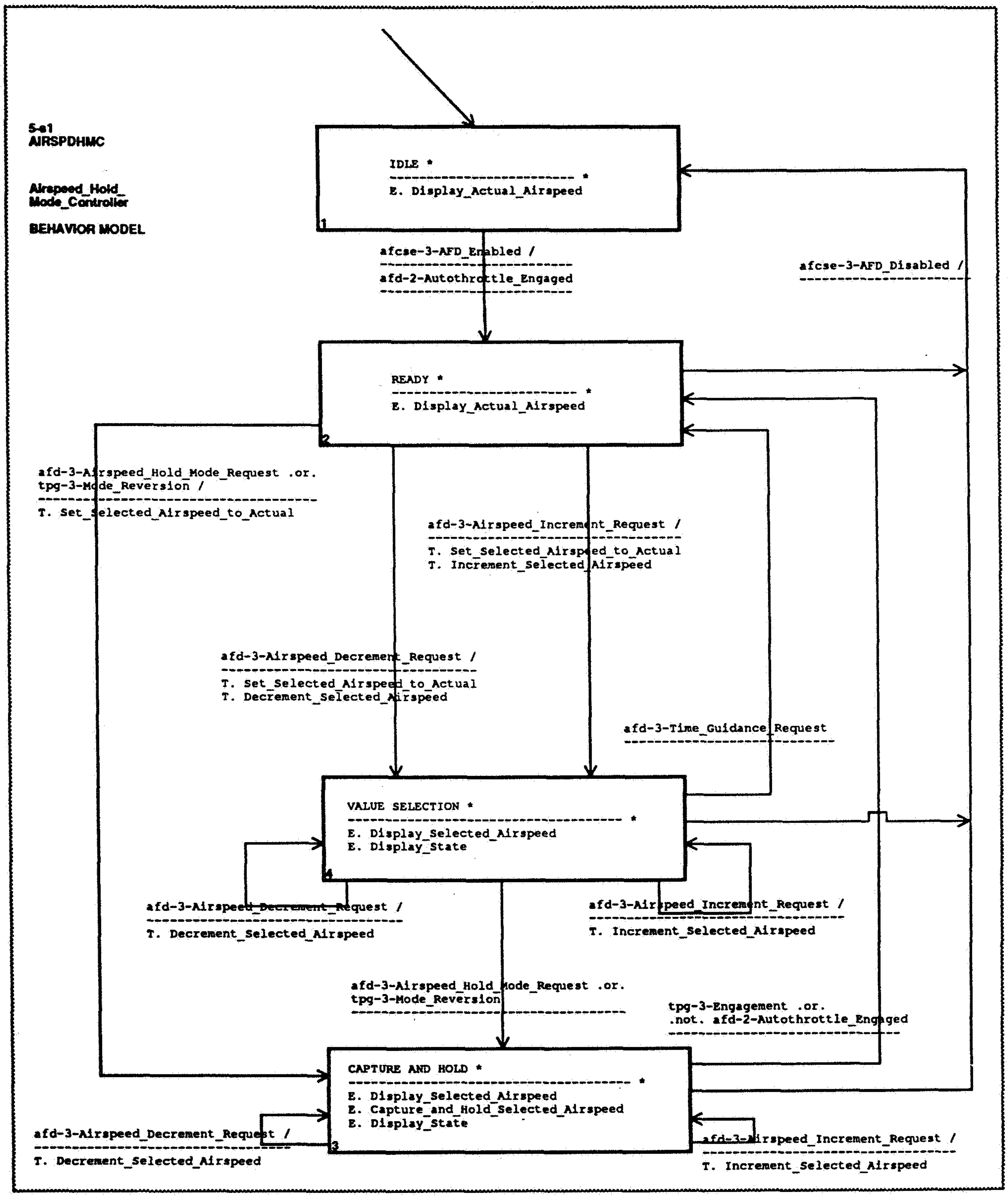


Figure 6.5 Behavior Model Diagram for AIRSPDHMC

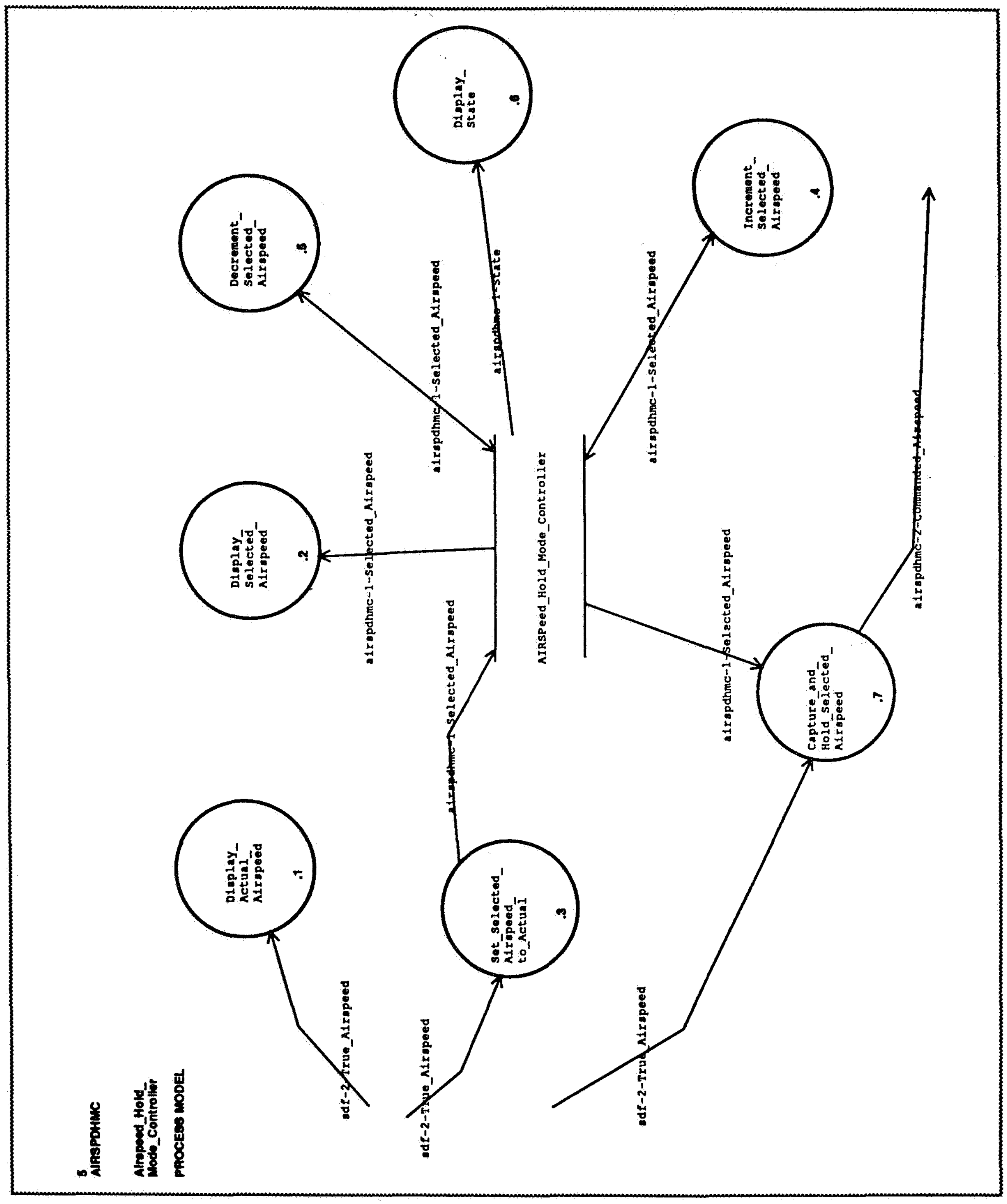


Figure 6.6 Process Model Diagram for AIRSPDHMC

6.1.7 ALTHMC--Altitude_Hold_Mode_Controller

(*anonymous object class*) The Altitude_Hold_Mode_Controller, when engaged, generates steering commands so as to capture and hold a pilot-selected altitude. It issues a vertical acceleration steering command.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Attribute:	althmc-1-Selected_Altitude
Derived Attribute:	althmc-2-Altitude_Difference
Derived Attribute:	althmc-2-Commanded_Vertical_Acceleration
Event:	althmc-3-Disengagement_Criteria_Triggered
Event:	althmc-3-Engagement_Criteria_Satisfied
Process:	Display_Actual_Altitude
Process:	Display_Selected_Altitude
Process:	Set_Selected_Altitude_to_Actual
Process:	Increment_Selected_Altitude
Process:	Decrement_Selected_Altitude
Process:	Capture_and_Hold_Selected_Altitude
Process:	Watch_Altitude_Difference
Process:	Display_State

Definitions

althmc-1-Selected_Altitude
(*attribute*)

This is the pilot-indicated altitude which shall be captured and held when the Altitude_Hold_Mode_Controller is engaged.

domn-Altitude_Above_Mean_Sea_Level.

althmc-2-Altitude_Difference
(*derived attribute*)

This is arithmetic difference between the actual and the select altitudes.

domn-Altitude_Above_Mean_Sea_Level.

althmc-2-Commanded_Vertical_Acceleration
(*derived attribute*)

This is the commanded vertical acceleration of the aircraft which shall be attained.

domn-Acceleration.

althmc-3-Disengagement_Criteria_Triggered
(*broadcast event*)

This event corresponds to a determination that the Altitude Hold Mode Controller must disengage, that is, release control of the aircraft.

althmc-3-Engagement_Criteria_Satisfied
(broadcast event)

This event corresponds to a determination that the Altitude Hold Mode Controller can successfully capture the Selected Altitude.

Display_Actual_Altitude
(process)

This process displays the Actual_Altitude of the aircraft.

Display_Selected_Altitude
(process)

This process displays the Selected_Altitude of the aircraft.

Set_Selected_Altitude_to_Actual
(process)

This process makes the Selected_Altitude attribute equal to the instantaneous Actual_Altitude.

Increment_Selected_Altitude
(process)

This process increases the current Selected_Altitude by 50 feet.

Decrement_Selected_Altitude
(process)

This process decreases the current Selected_Altitude by 50 feet.

Capture_and_Hold_Selected_Altitude
(process)

This process generates a vertical acceleration steering command so as to smoothly change the actual altitude of the aircraft to match the pilot-selected altitude.

Watch_Altitude_Difference
(process)

This process monitors the difference between the Actual_Altitude of the aircraft and the Selected_Altitude. At the moment the difference is small compared to the vertical speed of the aircraft, this process signals the Engagement_Criteria_Satisfied event, so that the selected altitude may be captured. At the moment the difference exceeds the maximum expected during altitude hold, the Disengagement_Criteria_Triggered event is signalled by this process, so that the Capture_and_Hold process can be disengaged.

Display_State
(process)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

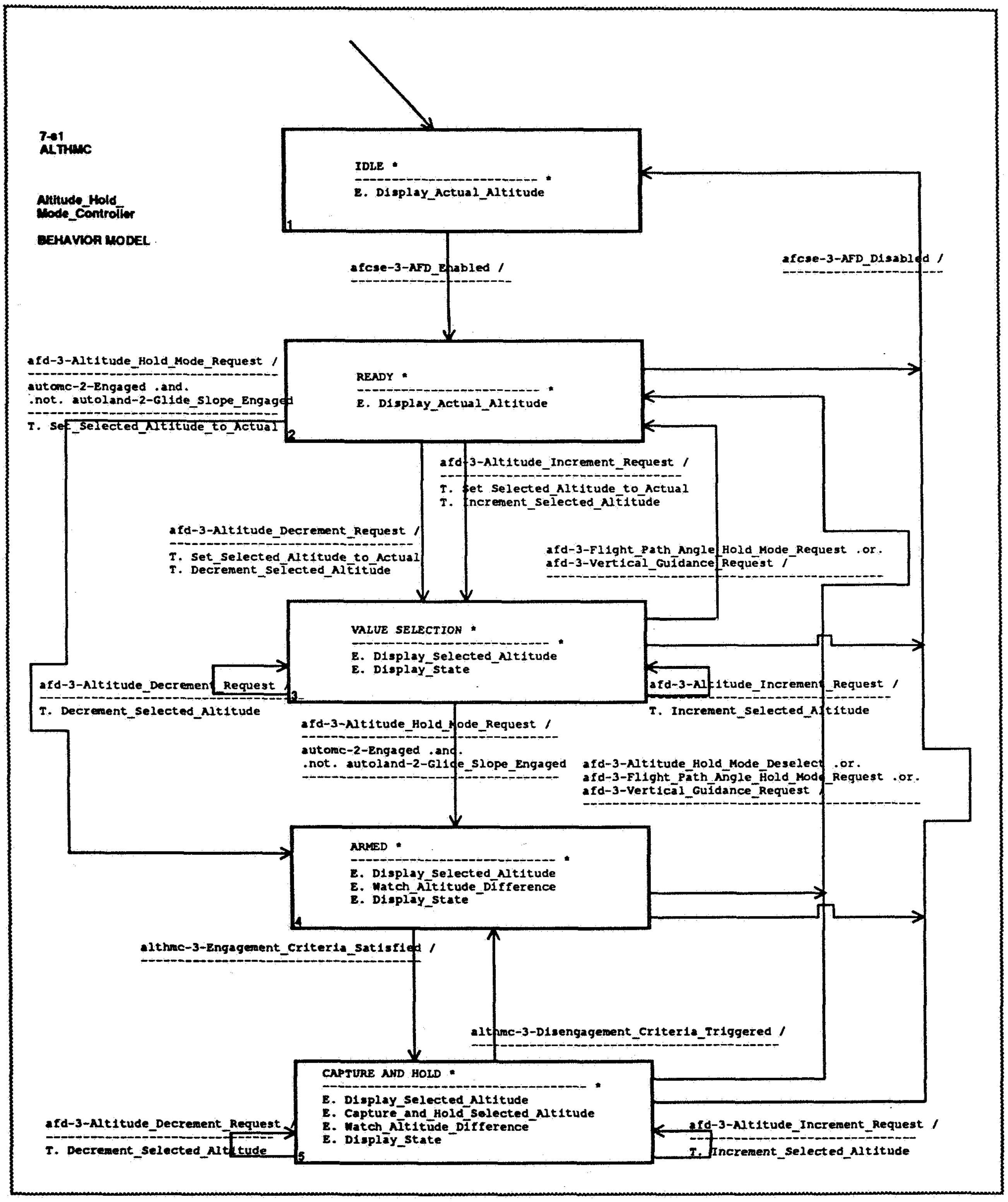


Figure 6.7 Behavior Model Diagram for ALTHMC

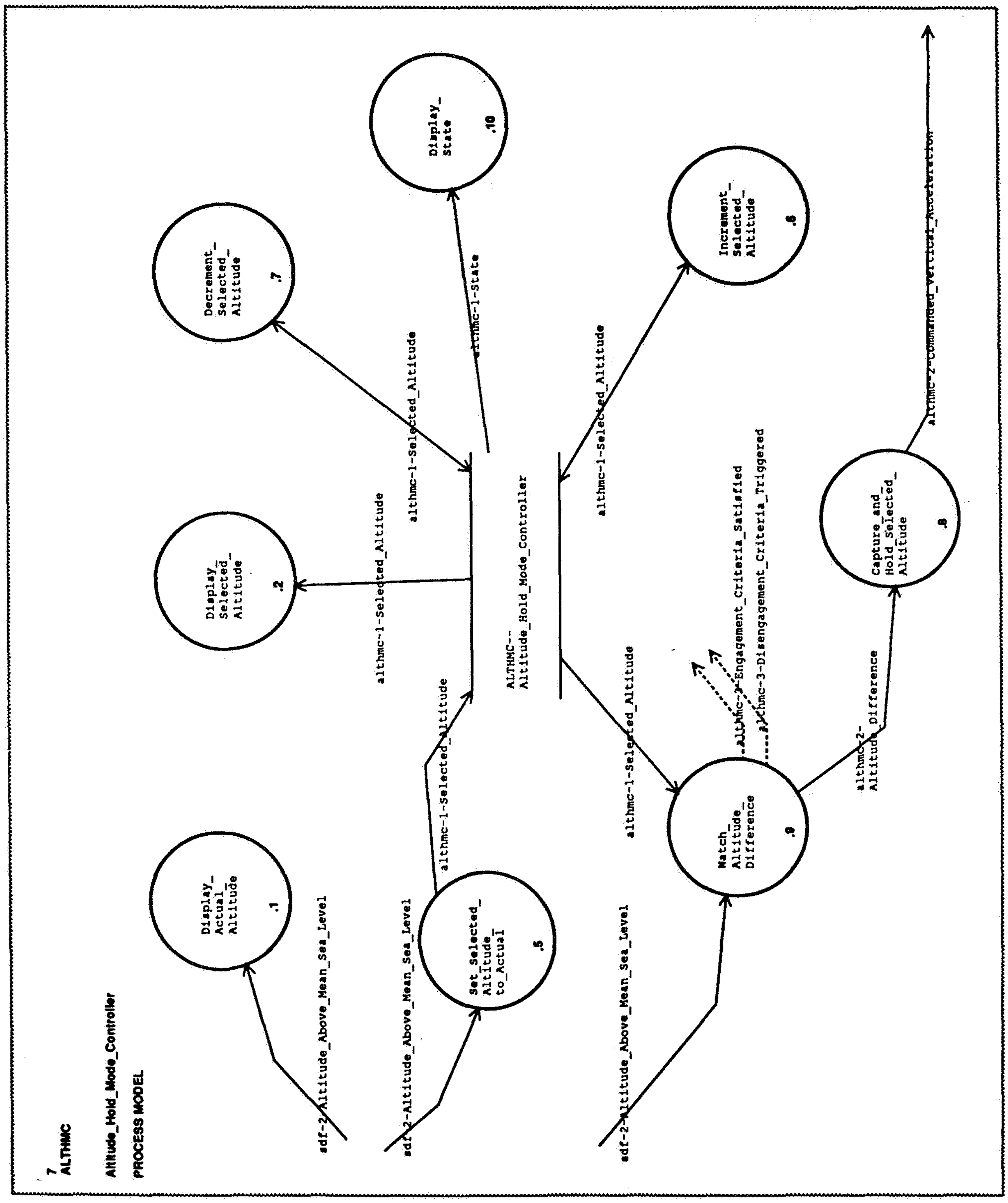


Figure 6.8 Process Model Diagram for ALTHMC

6.1.8 AP--Airport

(*regular object class*) An AirPort is an air traffic control area from which aircraft take off and at which aircraft land. That is, the definition here follows the ordinary usage.

Summary

Primary Identifier:	ap-1-Name
Attribute:	ap-1-Automd_Trml_Info_Svc_Radio_Freq
Attribute:	ap-1-Clearance_Radio_Frequency
Attribute:	ap-1-Ground_Control_Radio_Frequency
Attribute:	ap-1-Is_it_Usable
Attribute:	ap-1-Name
Attribute:	ap-1-Reference_Point_Latitude
Attribute:	ap-1-Reference_Point_Longitude
Attribute:	ap-1-Tower_Radio_Frequency

Definitions

ap-1-Automd_Trml_Info_Svc_Radio_Freq
(*attribute*)

This item is useful for the human pilot, and is otherwise unused in this analysis.

> domn-Radio_Frequency.

ap-1-Clearance_Radio_Frequency
(*attribute*)

This item is useful for the human pilot, and is otherwise unused in this analysis.

> domn-Radio_Frequency.

ap-1-Ground_Control_Radio_Frequency
(*attribute*)

This item is useful for the human pilot, and is otherwise unused in this analysis.

> domn-Radio_Frequency.

ap-1-Is_it_Usable
(*attribute*)

This indicates whether the aircraft is allowed to land at the Airport under normal conditions. This item determines how the airport is displayed to the human pilot.

> Type: Enumeration;
> Values: Operational, Display_Only;

ap-1-Name
(attribute)

The official id of the airport; not its common English name.

Type:	String;
Range:	exactly 4 upper case letters;

ap-1-Reference_Point_Latitude
(attribute)

This is the latitude of the official "location" of the airport as defined by a single point on the Earth's surface.

domn-Latitude.

ap-1-Reference_Point_Longitude
(attribute)

This is the longitude of the official "location" of the airport as defined by a single point on the Earth's surface.

domn-Longitude.

ap-1-Tower_Radio_Frequency
(attribute)

This item is useful for the human pilot, and is otherwise unused in this analysis.

domn-Radio_Frequency.

6.1.9 ATCWAR--ATC_Waypoint_onan_Airway_Route

(*regular object class*) A ATC_Waypoint_onan_Airway_Route arises out of the association of a particular ATC_Named_Waypint (ATCWPT) with a particular Published Airway Route (PAR). It captures the information relevant to such an association. (The same ATCWPT on a different PAR may be assigned different data values.)

"ATC_Waypoint_onan_Airway_Route" is a somewhat artificial term created for the sake of clarity in this analysis; in general usage, the term "ATC-Named Waypoint" may refer to both ATCWPTs and ATCWARs (where the meaning is clear from context). For that matter, the term "Waypoint" may refer to an ATCWPT or even an ATCWAR.

Summary

Primary Identifier:	atcwar-1-Route_Name-R10B + atcwar-1-Wpt_Name-R10A.
Attribute:	atcwar-1-Route_Name-R10B
Attribute:	atcwar-1-Wpt_Name-R10A

Definitions

atcwar-1-Route_Name-R10B

(*compound referential attribute*)

The id of the Published Airway Route which the named ATC Named Waypoint is on by means of this ATCWAR.

	par-1-Type
+	par-1-Number.

atcwar-1-Wpt_Name-R10A

(*referential attribute*)

The id of the ATC Named Waypoint which is on the named Published Airway Route by means of this ATCWAR.

atcwpt-1-Waypoint_Name-R17.

6.1.10 ATCWPT--ATC_Named_Waypoint

(regular object class) An ATC_Named_Waypoint is a Waypoint which is officially recognized, and which is documented on air navigation charts. Being an official entity, an ATCWPT is not modifiable by a pilot.

Summary

Primary Identifier:	atcwpt-1-Waypoint_Name-R17
Attribute:	atcwpt-1-Type-R18
Attribute:	atcwpt-1-Waypoint_Name-R17

Definitions

atcwpt-1-Type-R18

(subtype indication)

Type:	Subtype Indication;
Objects:	NAVAID, ISN;

atcwpt-1-Waypoint_Name-R17

(referential attribute)

wpt-1-Name.

6.1.11 AUTOLAND--Automatic_Landing_Controller

(*external, anonymous object class, no attributes*) This object class is a "pseudo external object class": treated as external until such time as this aspect of the system can be analyzed further. For the interim, this object class provides a source and sink for closely related events and data: those associated with landing and known to be required by objects already defined in this analysis. Further analysis may reveal that this single placeholder must become several object classes. Also, several of these pseudo external object classes might be split and joined.

Summary

Primary Identifier:	*none*
Derived Attribute:	autoland-2-Glide_Slope_Engaged
Derived Attribute:	autoland-2-Localizers_Engaged

Definitions

autoland-2-Glide_Slope_Engaged

(*derived attribute*)

domn-Boolean.

autoland-2-Localizers_Engaged

(*derived attribute*)

domn-Boolean.

6.1.12 AUTOMC--Auto_Modes_Controller

(*anonymous object class, no attributes*) The AUTOMC enables the engagement of the various "automatic flight modes".

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Derived Attribute:	automc-2-Auto_Modes_Engaged

Definitions

automc-2-Auto_Modes_Engaged
(*derived attribute*)

"True" while the AAMC has authorized the engagement of any of the automatic flight modes; "false" when that authorization has been retracted.

domn-Boolean.

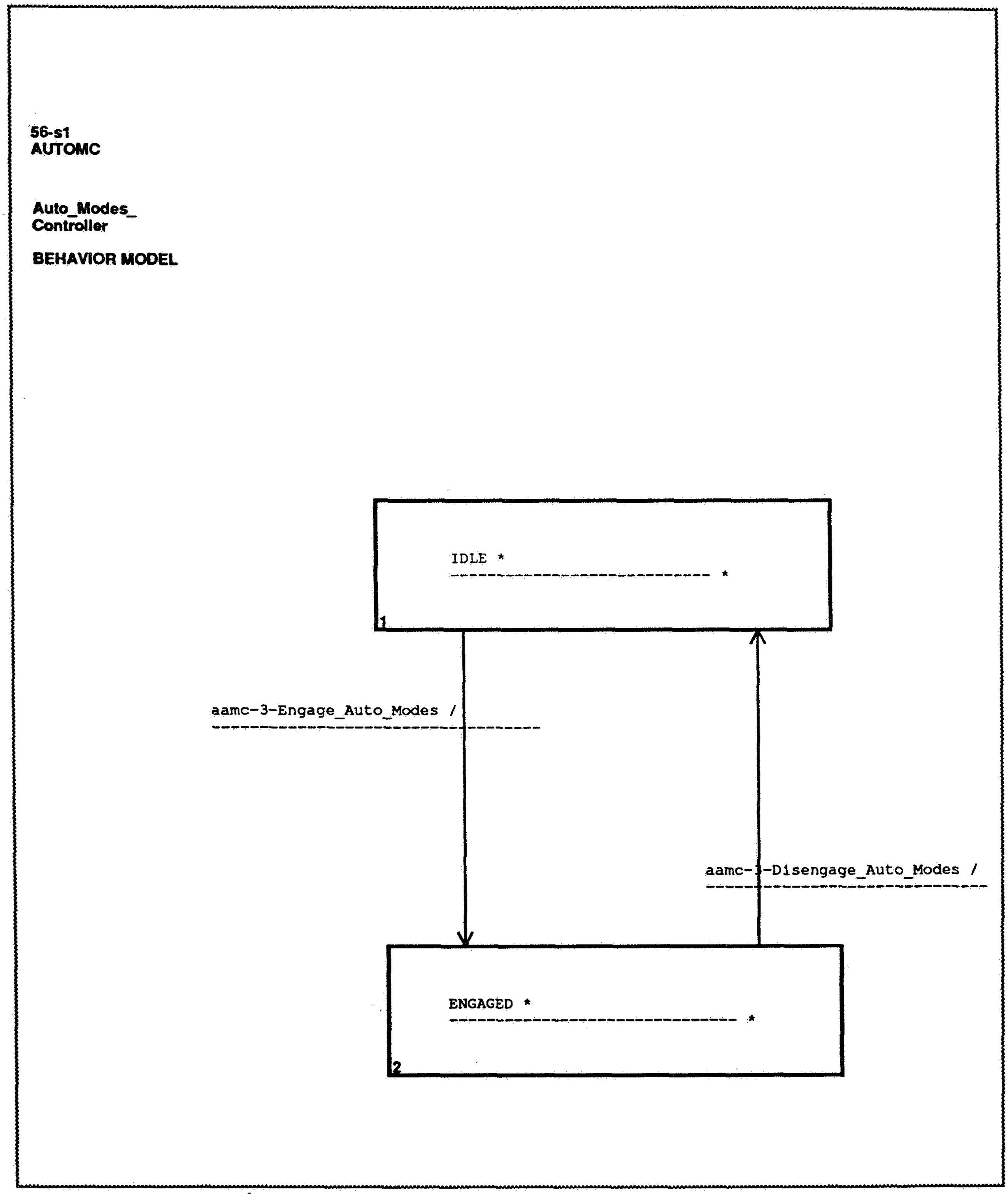


Figure 6.9 Behavior Model Diagram for AUTOMC

6.1.13 CR--Company_Route

(*regular object class*) A Company_Route is a pilot- or company-defined Flight_-Plan which is used frequently. Company_Routes aid the pilot by eliminating the need to "manually" enter the same sequence of Waypoints each time a frequently travelled path is desired. "Company Route" is a generally recognized term of the aviation community. Note: it is thought that Company_Routes generally begin and end at the edges of airport air traffic control areas.

Summary

Primary Identifier:	cr-1-Name.
Attribute:	cr-1-Name

Definitions

cr-1-Name
(*attribute*)

Type:	String;

6.1.14 CWFP--Constrained_Waypoint_ona_Flight_Plan

(*regular object class*) A Constrained_Waypoint_ona_Flight_Plan is a special kind of WFP used primarily for SID and STAR waypoints. A WFP of this type allows more than the usual, simple specification of an altitude or planned time of arrival; rather, altitude and airspeed constraints on a flight leg are recorded.

Summary

Primary Identifier:	cwfp-1-Flight_Plan_Name-R02
	+ cwfp-1-Waypoint_Name-R02
Attribute:	cwfp-1-Airspeed_Constraint_for_Wpt_Acq
Attribute:	cwfp-1-Airspeed_Value_for_Wpt_Acq
Attribute:	cwfp-1-Altitude_Constraint_at_Wpt
Attribute:	cwfp-1-Altitude_Value_at_Wpt
Attribute:	cwfp-1-Flight_Plan_Name-1-R02
Attribute:	cwfp-1-Waypoint_Name-1-R02

Definitions

cwfp-1-Airspeed_Constraint_for_Wpt_Acq
(*attribute*)

This constraint must be continually met during the whole of waypoint acquisition. It applies to the cwfp-1-Airspeed_Value_for_Wpt_Acq.

domn-Leg_Constraint.

cwfp-1-Airspeed_Value_for_Wpt_Acq
(*attribute*)

This defines the airspeed for the aircraft for the whole of waypoint acquisition. The attribute cwfp-1-Airspeed_Constraint_for_Wpt_Acq defines what the aircraft must do with this value.

domn-Airspeed.

cwfp-1-Altitude_Constraint_at_Wpt
(*attribute*)

This constraint must be met just by the time the aircraft is actually at the waypoint. It applies to the cwfp-1-Altitude_Value_at_Wpt.

domn-Point_Constraint.

cwfp-1-Altitude_Value_at_Wpt
(*attribute*)

This defines the altitude for the aircraft just when actually at the waypoint. The attribute cwfp-1-Altitude_Constraint_at_Wpt defines what the aircraft must do with this value.

domn-Altitude_Above_Mean_Sea_Level.

cwfp-1-Flight_Plan_Name-1-R02
(referential attribute)

wfp-1-Flight_Plan_Name-R01A.

cwfp-1-Waypoint_Name-1-R02
(referential attribute)

wfp-1-Waypoint_Name-R01B.

6.1.15 DA--Direct_Acquisition

(*regular object class*) A Direct_Acquisition is a Waypoint_ona_Flight_Plan which is acquired (flown to) directly (roughly, in a straight line) from the preceeding WFP.

Summary

Primary Identifier:	da-1-Flight_Plan_Name-R02
	+ da-1-Waypoint_Name-R02
Attribute:	da-1-Flight_Plan_Name-R02
Attribute:	da-1-Waypoint_Name-R02
Derived Attribute:	da-2-Distance_to_Tangent

Definitions

da-1-Flight_Plan_Name-R02

(*referential attribute*)

wfp-1-Flight_Plan_Name-R01A.

da-1-Waypoint_Name-R02

(*referential attribute*)

wfp-1-Waypoint_Name-R01B.

da-2-Distance_to_Tangent

(*derived attribute*)

Indicates the distance to the tangent point of the standard rate turn arc which is used to transition smoothly from the current leg to the next.

Type:	Numeric;
Range:	from 0 to <> nautical miles;

6.1.16 DEFMC--Default_Mode_Controller

(*anonymous object class, no attributes*) The Default Mode Controller controls the engagement and disengagement of the designated "default mode" (which is chosen by the AFD).

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Event:	defmc-3-Disengage_ACWS
Event:	defmc-3-Disengage_MANEL
Event:	defmc-3-Engage_ACWS
Event:	defmc-3-Engage_MANEL
Event:	defmc-3-Mode_Reversion

Definitions

defmc-3-Disengage_ACWS
(*broadcast event*)

This event corresponds to a determination that the ACWS must disengage.

defmc-3-Disengage_MANEL
(*broadcast event*)

This event corresponds to a determination that the MANEL must disengage.

defmc-3-Engage_ACWS
(*broadcast event*)

This event corresponds to a determination that the ACWS should engage.

defmc-3-Engage_MANEL
(*broadcast event*)

This event corresponds to a determination that the MANEL should engage.

defmc-3-Mode_Reversion
(*broadcast event*)

This event corresponds to the self-disengagement of the DEFMC. The event is generated to let other objects in the system react as they will.

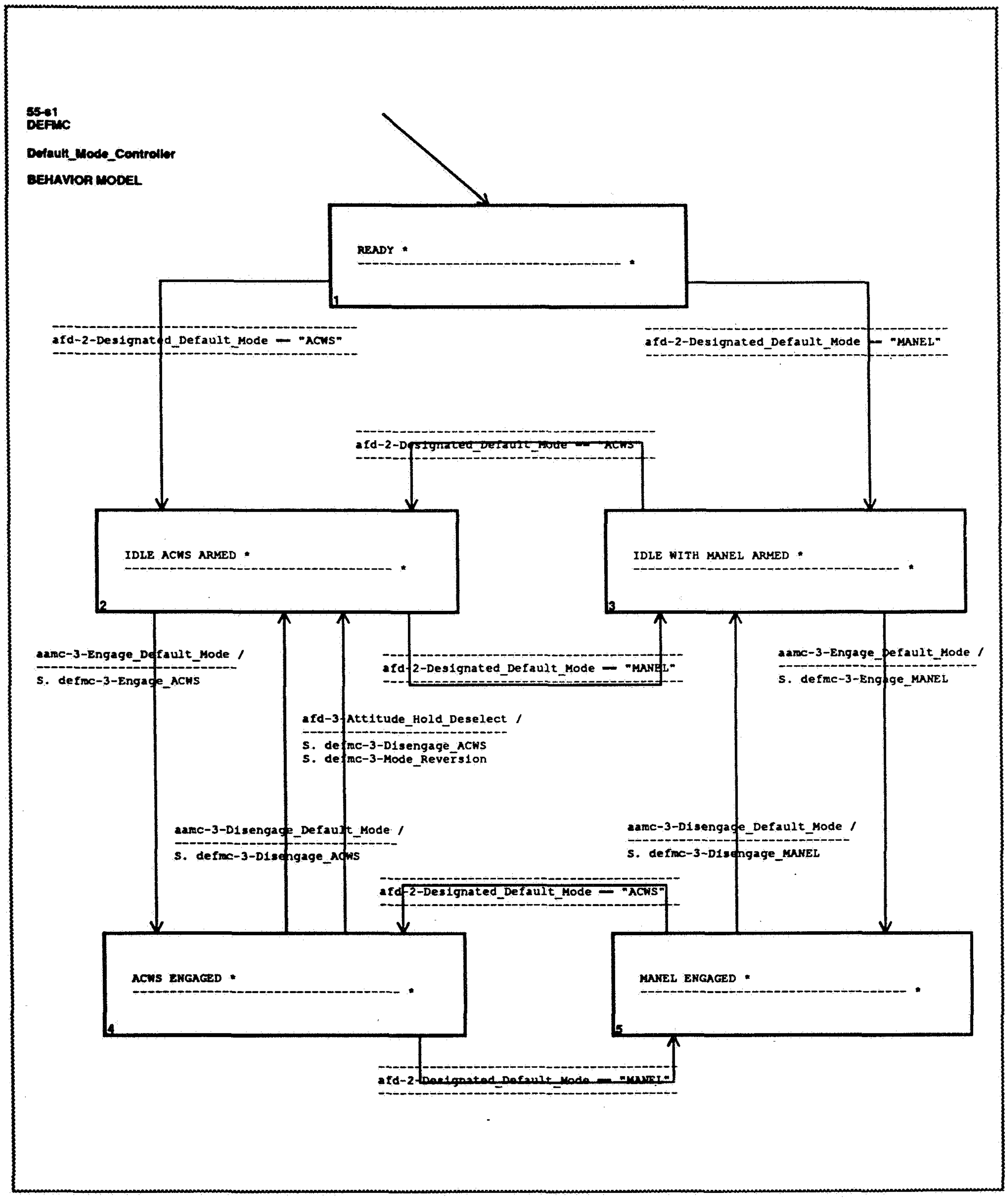


Figure 6.10 Behavior Model Diagram for DEFMC

6.1.17 DMEAA--DME_Arc_Acquisition

(*regular object class*) A DMEAA is a Waypoint_ona_Flight_Plan which is acquired (flown to) via an arc (with the aid of DME) from the preceeding WFP. The preceeding WFP is known as the "inbound waypoint" of the dme arc, while this WFP is known as the "outbound waypoint". This is why the method of acquiring this WFP is said to be via a dme arc: that is the path in space between the two WFPs.

Summary

Primary Identifier:	dmeaa-1-Flight_Plan_Name-R03
	+ dmeaa-1-Waypoint_Name-R03
Attribute:	dmeaa-1-Direction_of_Turn
Attribute:	dmeaa-1-Flight_Plan_Name-R03
Attribute:	dmeaa-1-Turn_Center_Wpt_Name-R04
Attribute:	dmeaa-1-Waypoint_Name-R03

Definitions

dmeaa-1-Direction_of_Turn
(*attribute*)

This attribute partially defines the arc acquisition path; without it, two different paths (together forming a circle about the Turn_Center) would be ambiguously defined.

Type:	Enumeration;
Values:	Clockwise, Counter Clockwise;

dmeaa-1-Flight_Plan_Name-R03
(*referential attribute*)

fp-1-Name.

dmeaa-1-Turn_Center_Wpt_Name-R04
(*referential attribute*)

This attribute names the center of the circle used for the DME Arc acquisition. To be valid, the distance from this Turn Center to the current WFP must be equal to the distance from this Turn Center to the previous WFP. (That is, the two radii must be equal.)

wpt-1-Name.

dmeaa-1-Waypoint_Name-R03
(*referential attribute*)

wpt-Name.

6.1.18 EC--Effector_Controller

(*external, anonymous object class, no attributes*) The Effector_Controller accepts requests to change certain "point mass" characteristics of the airplane (e.g., airspeed, bank, pitch, vertical acceleration). It translates these requests into one or more commands to the primary control effectors of the aircraft: aileron, elevator, rudder, stabilizer, engine (via throttle lever position).

Note that in this analysis, where there may typically exist multiple coupled surfaces (e.g., upper and lower rudders), the surfaces are taken together as (logically) a single effector.

Note: this object class is a "pseudo external object class": treated as external until such time as this aspect of the system can be analyzed further. For the interim, this object class provides a source and sink for closely related events and data. Further analysis may reveal that this single placeholder must become several object classes. Also, several of these pseudo external object classes might be split and joined.

Summary

Primary Identifier:	*none*
Derived Attribute:	ec-2-Commanded_Aileron_Position
Derived Attribute:	ec-2-Commanded_Elevator_Position
Derived Attribute:	ec-2-Commanded_Rudder_Position
Derived Attribute:	ec-2-Commanded_Stabilizer_Position
Derived Attribute:	ec-2-Commanded_Throttle_Position

Definitions

ec-2-Commanded_Aileron_Position
(*derived attribute*)

domn-Aileron_Position.

ec-2-Commanded_Elevator_Position
(*derived attribute*)

domn-Elevator_Position.

ec-2-Commanded_Rudder_Position
(*derived attribute*)

domn-Rudder_Position.

ec-2-Commanded_Stabilizer_Position
(*derived attribute*)

domn-Stabilizer_Position.

ec-2-Commanded_Throttle_Position
(*derived attribute*)

domn-Engine_Throttle_Position.

6.1.19 ELEV--Elevator

(*external, anonymous object class, no attributes*) The Elevator is one of the aerodynamic control effectors used to control the motion of the aircraft. All elevator surfaces are taken together here to be (logically) a single effector.

Summary

Primary Identifier: *none*
Derived Attribute: elev-2-Current_Position

Definitions

elev-2-Current_Position
(*derived attribute*)

This is the current sensed position of the elevator.

domn-Elevator_Position.

6.1.20 ENG--Engine

(external, anonymous object class) The Engine is one of the control effectors used to control the motion of the aircraft. All the engines are taken together as a single entity.

Summary

Primary Identifier:	*none*
Derived Attribute:	eng-2-Current_Engine_Pressure_Ratio
Derived Attribute:	eng-2-Current_Exhaust_Gas_Temperature
Derived Attribute:	eng-2-Current_N1_RPM
Derived Attribute:	eng-2-Current_N2_RPM
Derived Attribute:	eng-2-Current_Throttle_Position

Definitions

eng-2-Current_Engine_Pressure_Ratio
(derived attribute)

domn-Engine_Pressure_Ratio.

eng-2-Current_Exhaust_Gas_Temperature
(derived attribute)

domn-Exhaust_Gas_Temperature.

eng-2-Current_N1_RPM
(derived attribute)

domn-Engine_RPM.

eng-2-Current_N2_RPM
(derived attribute)

domn-Engine_RPM.

eng-2-Current_Throttle_Position
(derived attribute)

domn-Engine_Throttle_Position.

6.1.21 FFD--Forward_Flight_Deck_Crew

(*external, anonymous object class, no attributes*) The Forward_Flight_Deck_Crew consists of the safety pilots who fly the aircaft in the forward flight deck.

Summary

Primary Identifier:	*none*
Derived Attribute:	ffd-2-AFD_Designated
Derived Attribute:	ffd-2-Column_Position
Derived Attribute:	ffd-2-FFD_Designated
Derived Attribute:	ffd-2-Rudder_Pedal_Position
Derived Attribute:	ffd-2-Wheel_Position

Definitions

ffd-2-AFD_Designated
(*derived attribute*)

"True" when the Forward Flight Deck Crew has selected Aft Flight Deck control of the aircraft.

domn-Boolean.

ffd-2-Column_Position
(*derived attribute*)

The control input which indicates the desired Pitch Angle of the aircraft.

domn-Column_Position.

ffd-2-FFD_Designated
(*derived attribute*)

"True" when the Forward Flight Deck Crew has selected Forward Flight Deck control of the aircraft.

domn-Boolean.

ffd-2-Rudder_Pedal_Position
(*derived attribute*)

The control input which indicates the desired Yaw Angle of the aircraft.

domn-Rudder_Pedal_Position.

ffd-2-Wheel_Position
(*derived attribute*)

The control input which indicates the desired Bank Angle of the aircraft.

domn-Wheel_Position.

6.1.22 FFDCWS--Fwd_Flight_Deck_Control_Wheel_Steering_Mode_-Controller

(*anonymous object class*) The Forward_Flight_Deck_Control_Wheel_Steering_-Mode_Controller, while engaged, maintains the attitude of the aircraft by issuing commands to the Effector_Controller. The FFDCWS allows the FFD_-Crew to change the attitude-being-maintained by responding to wheel/column inputs (or equivalent control inputs) while engaged.

Summary

Primary Identifier:	*none*
Alias:	ACWS Mode Controller
Behavior Model:	*present*
Process Model:	*present*
Attribute:	ffdcws-1-Selected_Bank_Angle
Attribute:	ffdcws-1-Selected_Pitch_Angle
Attribute:	ffdcws-1-Selected_Yaw_Angle
Derived Attribute:	ffdcws-2-Commanded_Bank_Angle
Derived Attribute:	ffdcws-2-Commanded_Pitch_Angle
Derived Attribute:	ffdcws-2-Commanded_Yaw_Angle
Derived Attribute:	ffdcws-2-Forward_Flight_Deck_CWS_Engaged
Process:	Set_Selected_Attitude_to_Actual
Process:	Maintain_Attitude_of_Aircraft

Definitions

ffdcws-1-Selected_Bank_Angle
(*attribute*)

domn-Bank_Angle.

ffdcws-1-Selected_Pitch_Angle
(*attribute*)

domn-Pitch_Angle.

ffdcws-1-Selected_Yaw_Angle
(*attribute*)

domn-Yaw_Angle.

ffdcws-2-Commanded_Bank_Angle
(*derived attribute*)

domn-Bank_Angle.

ffdcws-2-Commanded_Pitch_Angle
(*derived attribute*)

domn-Pitch_Angle.

ffdcws-2-Commanded_Yaw_Angle
(*derived attribute*)

domn-Yaw_Angle.

ffdcws-2-Forward_Flight_Deck_CWS_Engaged

(derived attribute)

"True" while the FFDCWS is engaged (i.e., controlling the aircraft so as to maintain an attitude); "false" when disengaged.

domn-Boolean.

Set_Selected_Attitude_to_Actual

(process)

This process makes the selected attitude (pitch, bank, yaw) of the aircraft equal to the instantaneous actual attitude.

Maintain_Attitude_of_Aircraft

(process)

This process commands the Effector_Controller to maintain the selected attitude (pitch, bank, yaw) of the aircraft, while at the same time allowing the Aft Flight Deck Crew to modify this selected attitude via control inputs.

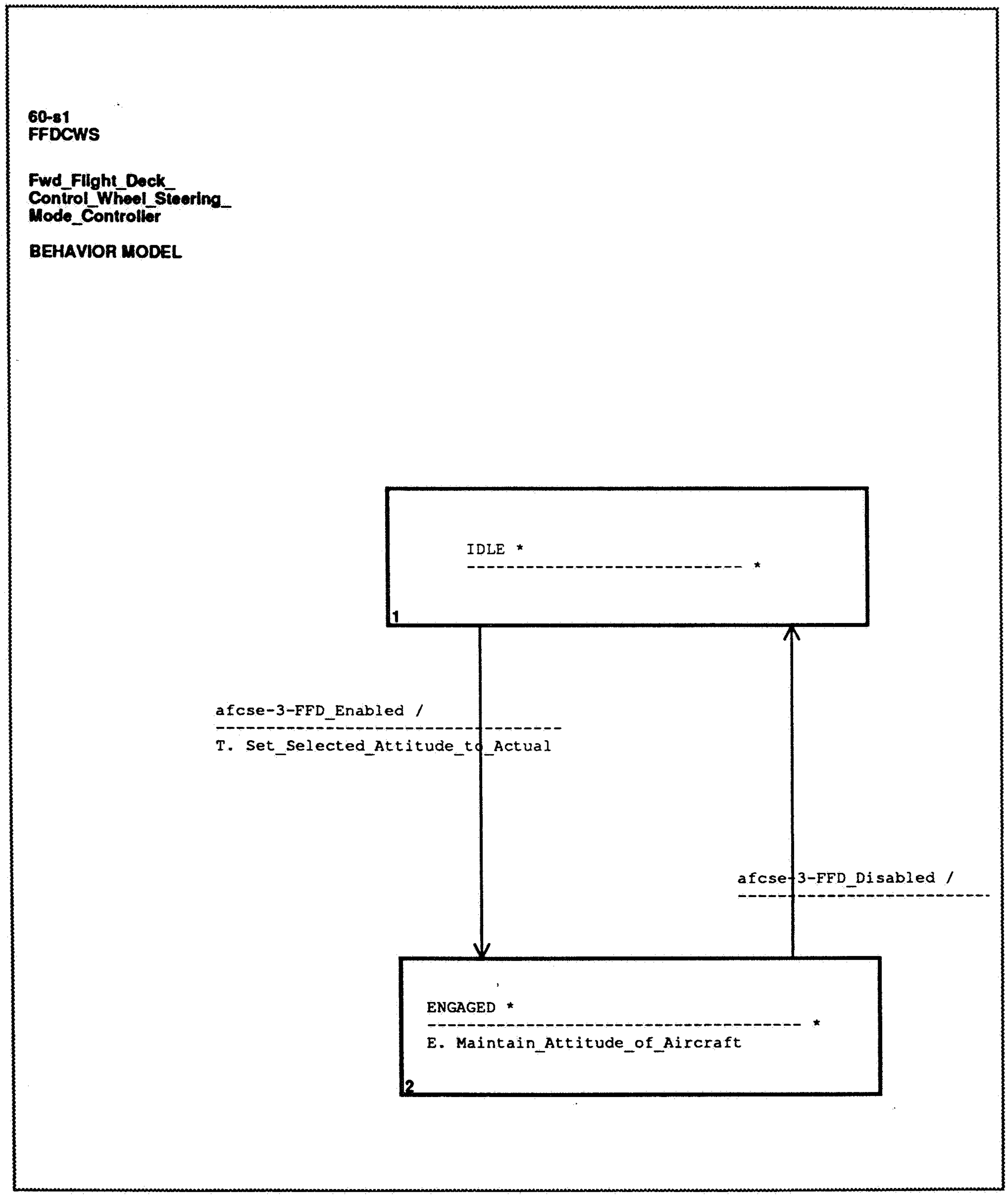


Figure 6.11 Behavior Model Diagram for FFDCWS

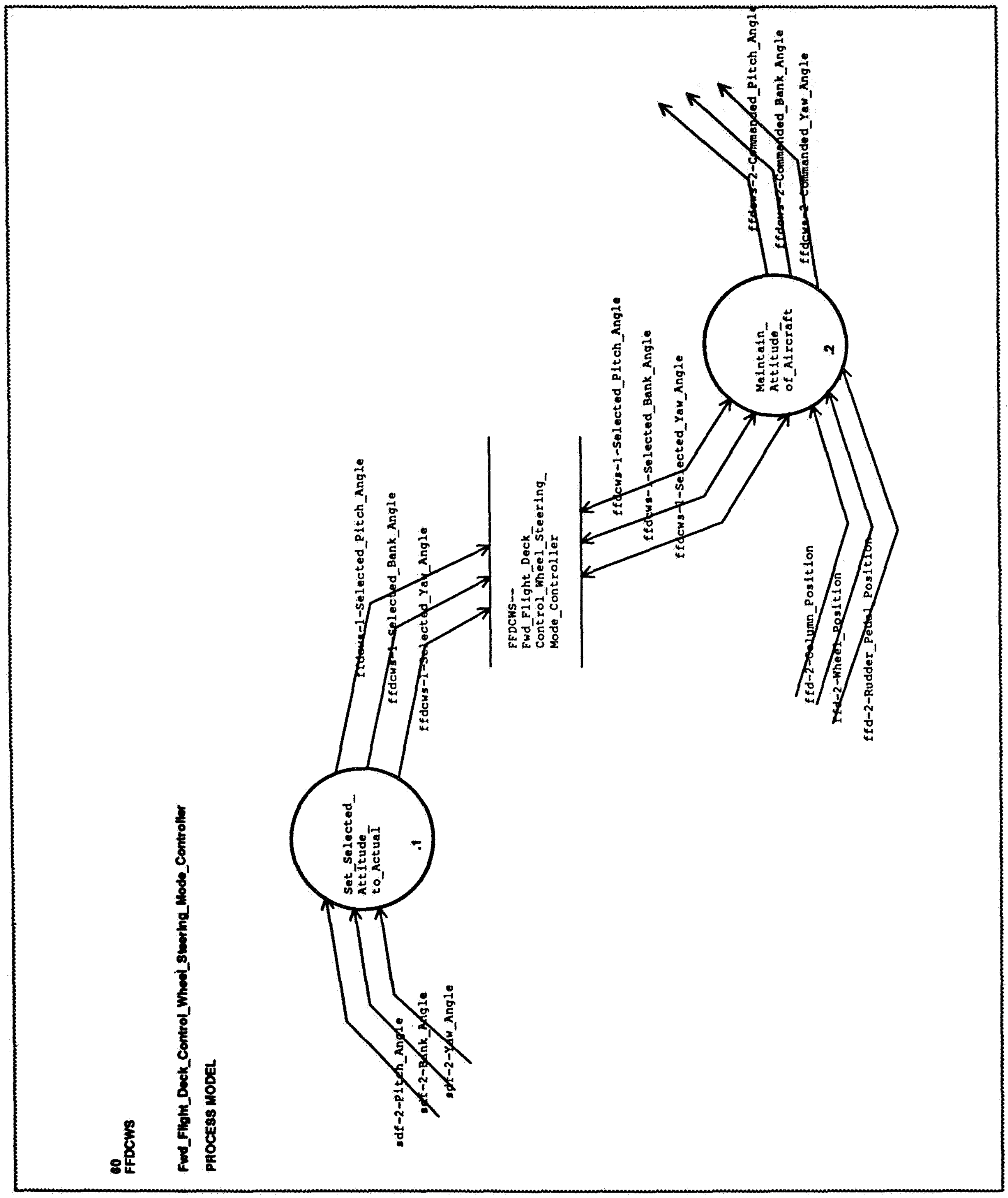


Figure 6.12 Process Model Diagram for FFDCWS

6.1.23 FP--Flight_Plan

(*regular object class*) A Flight_Plan is a scheduled path through space for an aircraft to follow. The scheduled path may be specified in two, three or four dimensions. When fewer than four dimensions are specified, the remaining dimensions are unconstrained and are determined by the pilot or system "on the fly" during flight. Different parts of the path may be specified in different numbers of dimensions. The path may contain so-called "discontinuities," which arise due to modifications which delete part of the path. (The discontinuity exists until and unless the end and start points are explicitly "joined.")

The AFD Crew specifies a Flight_Plan by listing an ordered sequence of Waypoints, possibly creating "pilot-defined" Waypoints in the process, but generally referencing pre-existing Waypoints. Whole strings of Waypoints may be specified in the creation process by referencing SIDs, STARs, Company Routes and/or Published Airway Routes (which themselves are ordered sequences of Waypoints). A list of Waypoints without additional information determines a two-dimensional Flight Plan. Additional constraints may be placed on individual Waypoints on the Flight Plan, such as specifying an Altitude (to form a 3-D Waypoint) and possibly a Planned Time of Arrival (to form a 4-D Waypoint); alternatively, SID/STAR-like constraints may be placed on a Waypoint (or such constraints copied from pre-existing SIDs and STARs may be modified).

A "Flight Plan" as commonly thought of in the aviation community begins at a Runway at one Airport and ends at another Runway at another Airport. However, this is the ideal, complete flight plan. The intermediate stages that exist during creation and editing are also FP--Flight_Plans, strictly speaking (merely incomplete ones).

Summary

Primary Identifier:	fp-1-Name.
Process Model:	*present*
Attribute:	fp-1-Name
Derived Attribute:	fp-2-Horizontal_Guidance_Possible
Derived Attribute:	fp-2-Time_Guidance_Possible
Derived Attribute:	fp-2-Vertical_Guidance_Possible
Process:	Modify_Provisional_Flight_Plan

Definitions

fp-1-Name
(*attribute*)

Note that only zero, one or two Flight_Plans exist at any time.

Type:	Enumeration;
Values:	Active, Provisional;

fp-2-Horizontal_Guidance_Possible
(derived attribute)

"True" when 3 or more 4-D Waypoints exist in the Provisional Flight_-Plan without discontinuities; "False" otherwise.

domn-Boolean.

fp-2-Time_Guidance_Possible
(derived attribute)

"True" when 3 or more 4-D Waypoints exist in the Provisional Flight_-Plan without discontinuities; "False" otherwise.

domn-Boolean.

fp-2-Vertical_Guidance_Possible
(derived attribute)

"True" when 3 or more 4-D Waypoints exist in the Provisional Flight_-Plan without discontinuities; "False" otherwise.

domn-Boolean.

Modify_Provisional_Flight_Plan
(process)

This process allows the insertion and deletion of Waypoints in the provisional Flight Plan. This process allows the pilot to insert whole strings of Waypoints into the provisional flight plan via reference to route entry/exit Waypoints. Any route known by the system can be used for this purpose: Published Airway Routes, Company Routes, as well as SIDs and STARs.

Deletion of a Waypoint in the middle of a Flight Plan shall create a "discontinuity" in the Flight Plan.

This process allows two Waypoints in the provisional Flight Plan to be "joined," wherein all Waypoints in between the two (if any) are deleted, and no discontinuity arises. (This is used to resolve a discontinuity.)

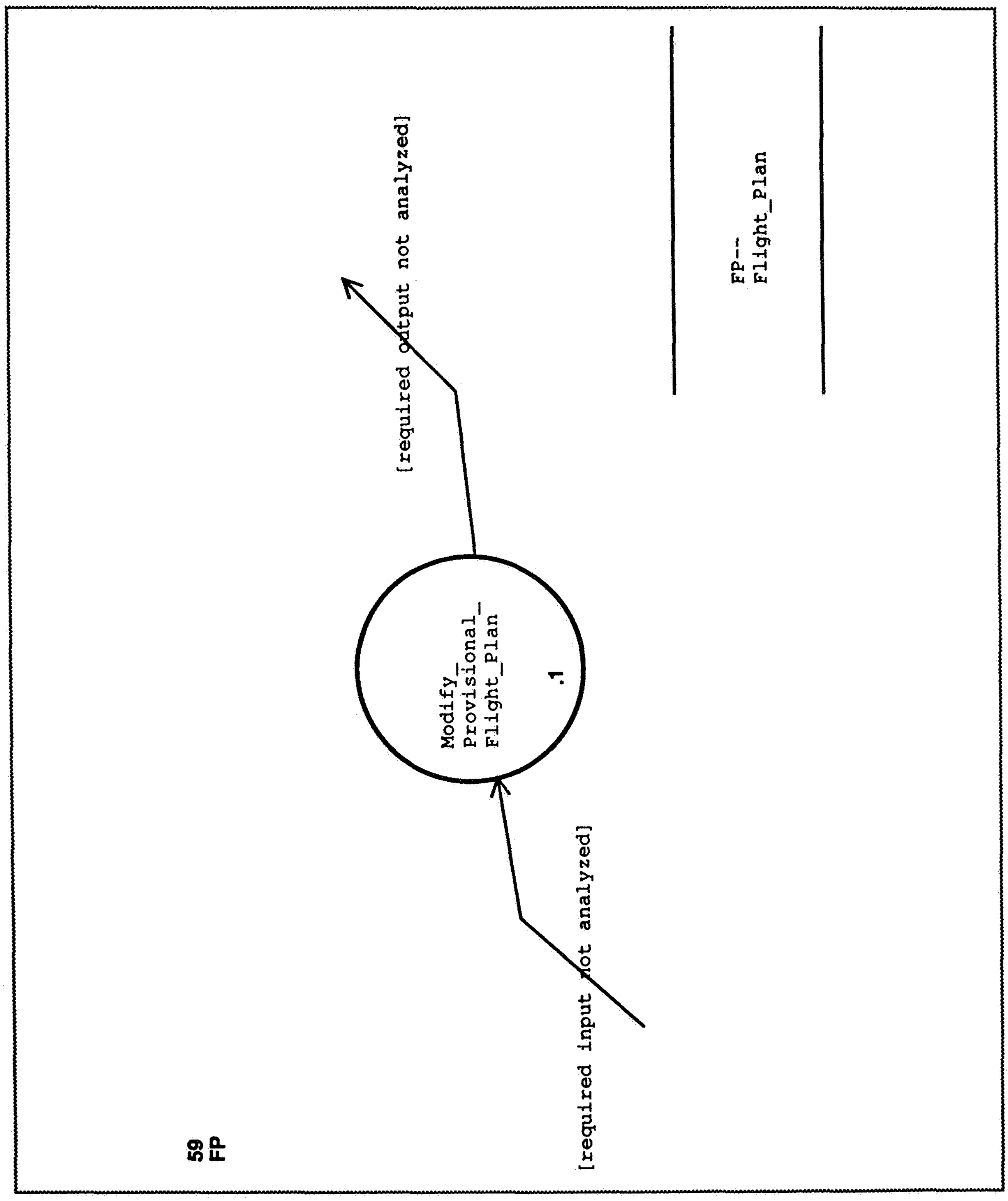


Figure 6.13 Process Model Diagram for FP

6.1.24 FPAHMC--Flight_Path_Angle_Hold_Mode_Controller

(*anonymous object class*) The Flight_Path_Angle_Hold_Mode_Controller, when engaged, generates steering commands so as to capture and hold a pilot-selected flight path angle. It issues a vertical acceleration steering command to the Effector_Controller.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Attribute:	fpahmc-1-Selected_Flight_Path_Angle
Derived Attribute:	fpahmc-2-Commanded_Vertical_Acceleration
Process:	Display_Actual_Flight_Path_Angle
Process:	Display_Selected_Flight_Path_Angle
Process:	Set_Selected_Flight_Path_Angle_to_Actual
Process:	Increment_Selected_Flight_Path_Angle
Process:	Decrement_Selected_Flight_Path_Angle
Process:	Display_State
Process:	Capture_and_Hold_Selected_Flight_Path_Angle

Definitions

fpahmc-1-Selected_Flight_Path_Angle

(*attribute*)

domn-Flight_Path_Angle.

fpahmc-2-Commanded_Vertical_Acceleration

(*derived attribute*)

domn-Acceleration.

Display_Actual_Flight_Path_Angle

(*process*)

This process displays the Actual_Flight_Path_Angle of the aircraft.

Display_Selected_Flight_Path_Angle

(*process*)

This process displays the Selected_Flight_Path_Angle.

Set_Selected_Flight_Path_Angle_to_Actual

(*process*)

This process makes the Selected_Flight_Path_Angle attribute equal to the instantaneous Actual_Flight_Path_Angle.

Increment_Selected_Flight_Path_Angle

(*process*)

This process increases the current Selected_Flight_Path_Angle by 0.1 degrees.

Decrement_Selected_Flight_Path_Angle

(process)

This process decreases the current Selected_Flight_Path_Angle by 0.1 degrees.

Display_State

(process)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

Capture_and_Hold_Selected_Flight_Path_Angle

(process)

This process generates a vertical acceleration steering command so as to smoothly change the Actual Flight Path Angle of the aircraft to match the Selected Flight Path Angle.

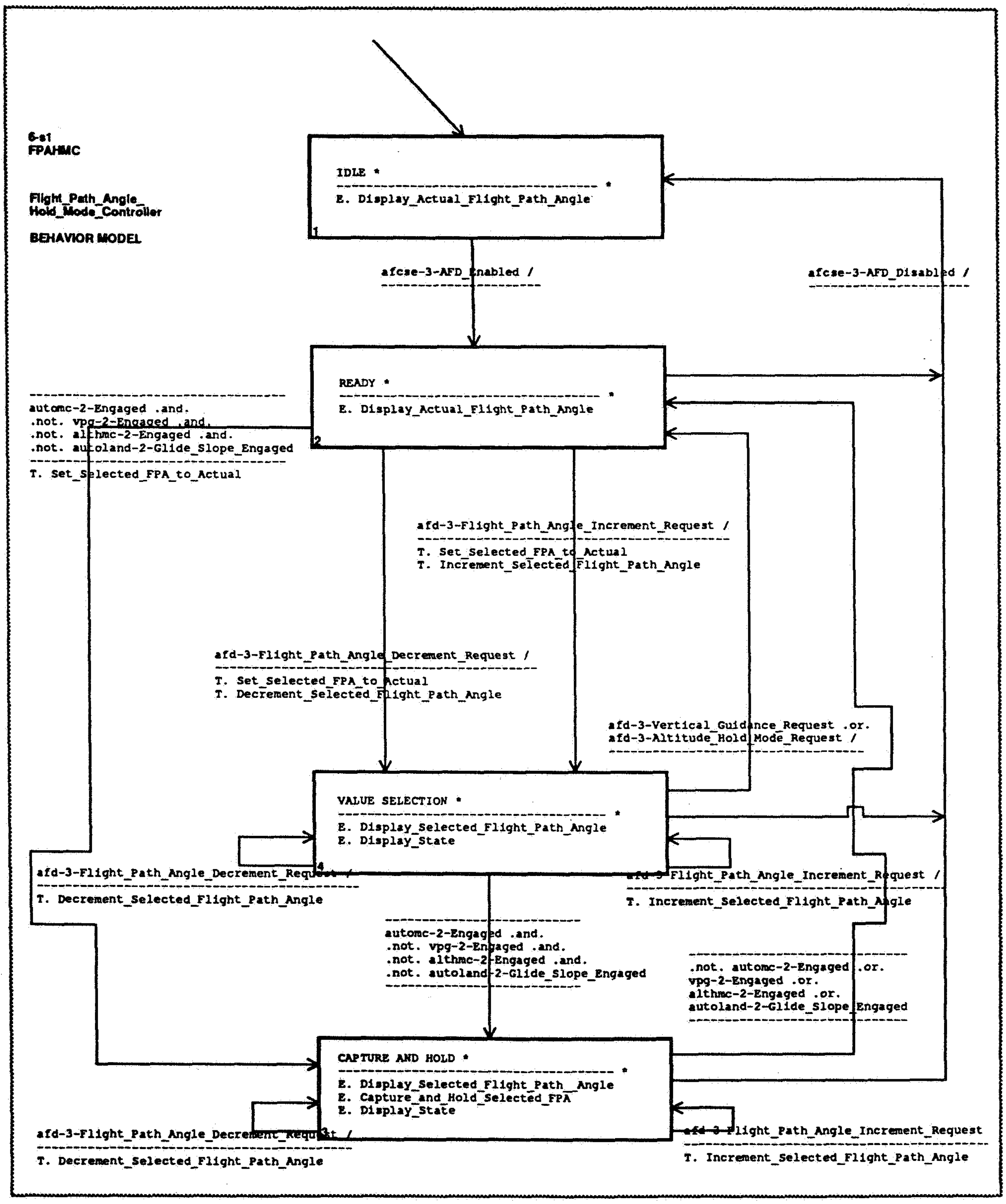


Figure 6.14 Behavior Model Diagram for FPAHMC

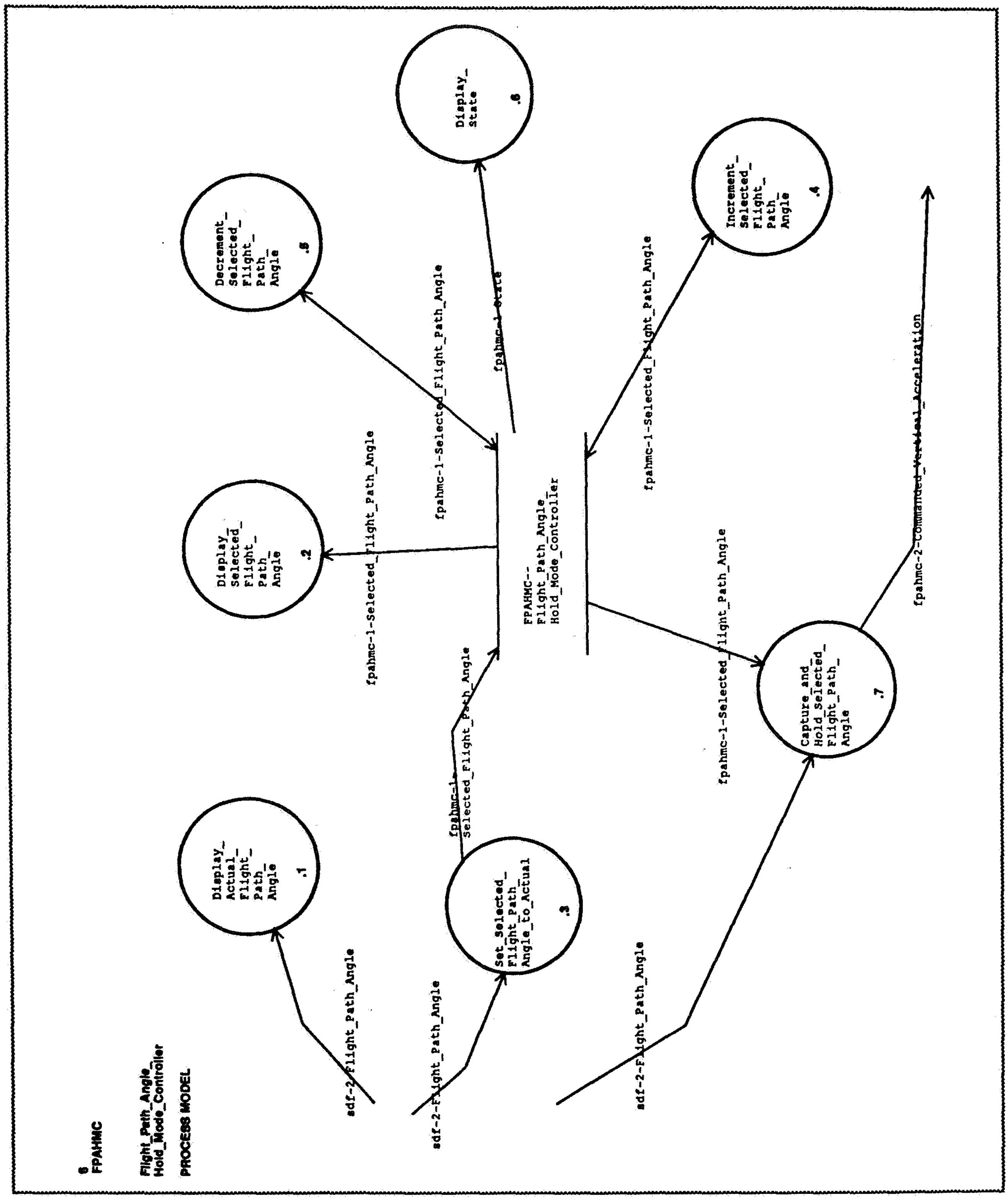


Figure 6.15 Process Model Diagram for FPAHMC

6.1.25 GTAHMC--Ground_Track_Angle_Hold_Mode_Controller

(*anonymous object class*) The Ground_Track_Angle_Hold_Mode_Controller, when engaged, generates steering commands so as to capture and hold a pilot-selected ground track angle. It issues a bank angle steering command to the Effector_Controller.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Attribute:	gtahmc-1-Selected_Ground_Track_Angle
Derived Attribute:	gtahmc-2-Commanded_Bank_Angle
Process:	Display_Actual_Ground_Track_Angle
Process:	Display_Selected_Ground_Track_Angle
Process:	Set_Selected_Ground_Track_Angle_to_Actual
Process:	Increment_Selected_Ground_Track_Angle
Process:	Decrement_Selected_Ground_Track_Angle
Process:	Capture_and_Hold_Selected_Ground_Track_Angle
Process:	Display_State

Definitions

gtahmc-1-Selected_Ground_Track_Angle
(*attribute*)

domn-Ground_Track_Angle.

gtahmc-2-Commanded_Bank_Angle
(*derived attribute*)

domn-Bank_Angle.

Display_Actual_Ground_Track_Angle
(*process*)

This process displays the Actual_Ground_Track_Angle of the aircraft.

Display_Selected_Ground_Track_Angle
(*process*)

This process displays the Selected_Ground_Track_Angle of the aircraft.

Set_Selected_Ground_Track_Angle_to_Actual
(*process*)

This process makes the Selected_Ground_Track_Angle attribute equal to the instantaneous Actual_Ground_Track_Angle.

Increment_Selected_Ground_Track_Angle
(*process*)

This process increases the current Selected_Ground_Track_Angle by one degree.

C-2

Decrement_Selected_Ground_Track_Angle

(process)

This process decreases the current Selected_Ground_Track_Angle by one degree.

Capture_and_Hold_Selected_Ground_Track_Angle

(process)

This process generates a bank angle steering command so as to smoothly change the Actual_Ground_Track_Angle of the aircraft to match the Selected_Ground_Track_Angle.

Display_State

(process)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

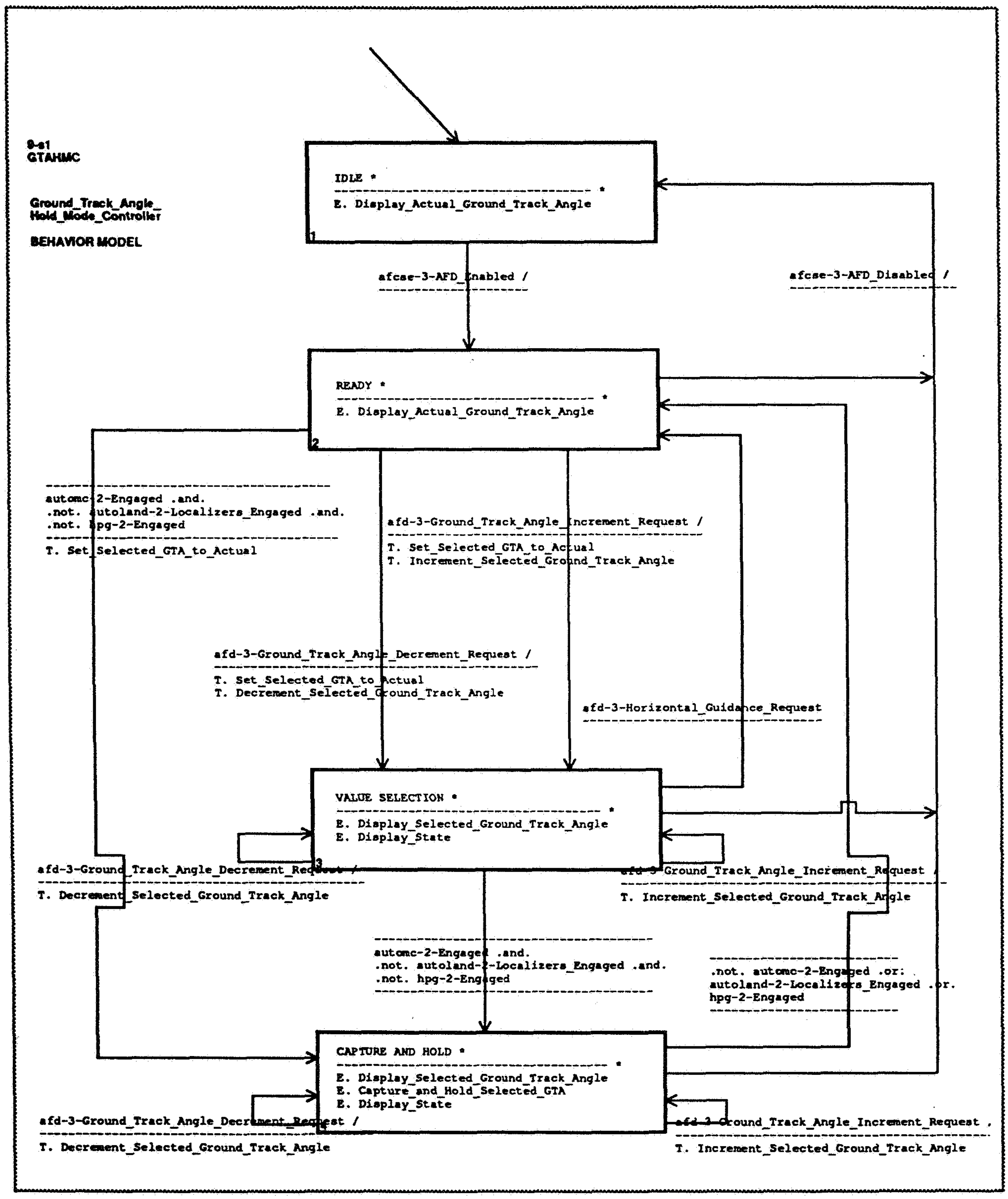


Figure 6.16 Behavior Model Diagram for GTAHMC

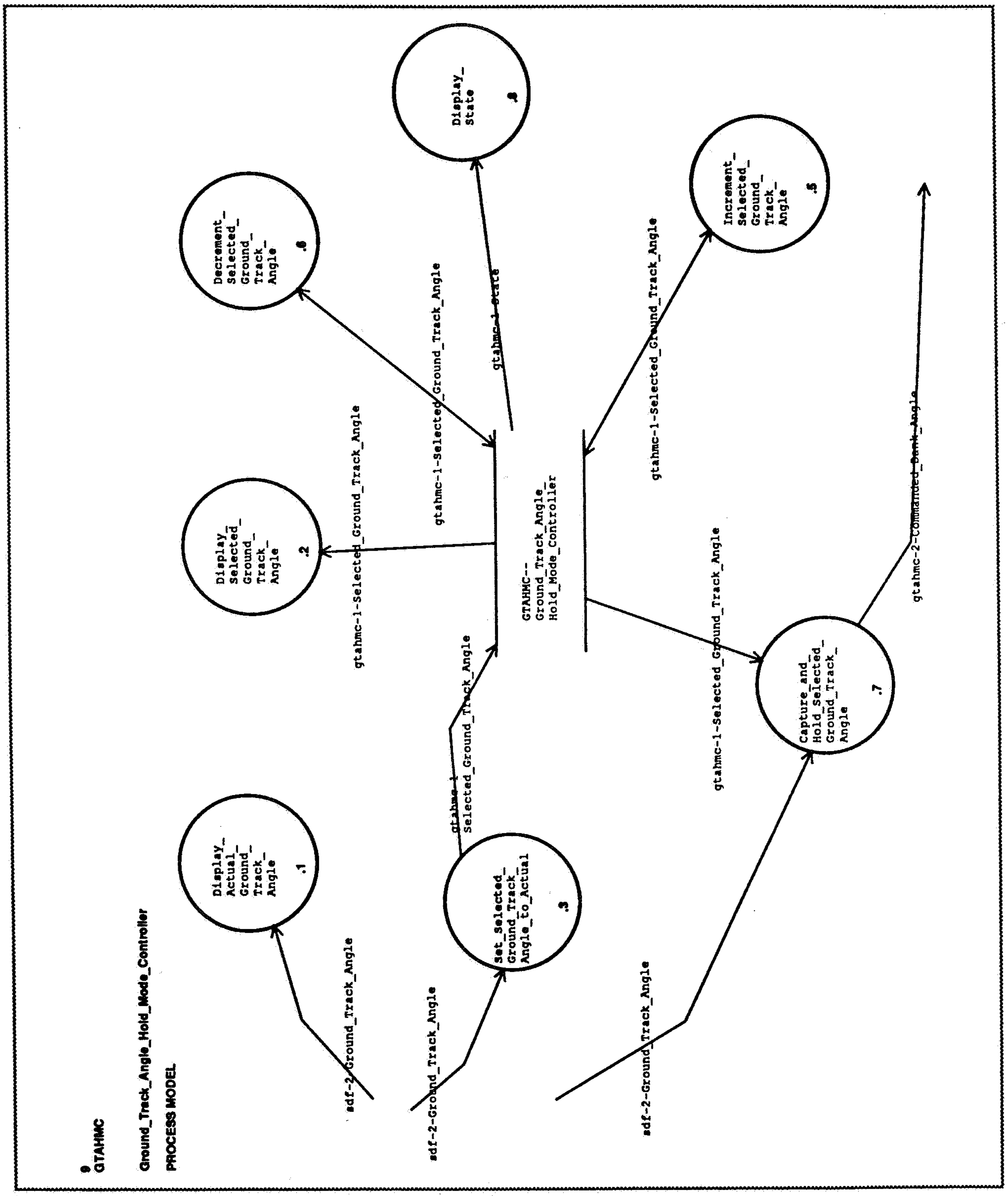


Figure 6.17 Process Model Diagram for GTAHMC

6.1.26 HPG--Horizontal_Path_Guide

(*anonymous object class, no attributes*) The HPG, while engaged, generates steering commands that direct the aircraft to, and maintain it on, the horizontal projection of the Flight_Plan. The HPG assumes the aircraft is near the leg between the first two waypoints of the flight plan.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Derived Attribute:	hpg-2-Commanded_Bank_Angle
Derived Attribute:	hpg-2-Engaged
Event:	hpg-3-Disengagement_Criteria_Triggered
Event:	hpg-3-Engagement_Criteria_Satisfied
Event:	hpg-3-Mode_Reversion
Event:	hpg-3-Path_Convergence
Event:	hpg-3-Path_Divergence
Event:	hpg-3-Waypoint_Acquired
Process:	Display_State
Process:	Evaluate_Engagement_Criteria
Process:	Evaluate_Disengagement_Criteria
Process:	Watch_for_Path_Divergence
Process:	Watch_for_Path_Convergence
Process:	Fly_Horiz_Comp_of_Active_Flight_Plan

Definitions

hpg-2-Commanded_Bank_Angle
(*derived attribute*)

This is the commanded bank angle which shall be attained.

domn-Bank_Angle.

hpg-2-Engaged
(*derived attribute*)

"True" when the HPG is engaged; false otherwise.

domn-Boolean.

hpg-3-Disengagement_Criteria_Triggered
(*broadcast event*)

This event corresponds to a determination that the HPG must disengage, that is, release control of the aircraft.

hpg-3-Engagement_Criteria_Satisfied
(*broadcast event*)

This event corresponds to a determination that the HPG can successfully capture the horizontal component of the flight plan.

hpg-3-Mode_Reversion
(broadcast event)

This event corresponds to the self-disengagement of the HPG. The event is generated to let others in the system react as they will.

hpg-3-Path_Convergence
(broadcast event)

This event corresponds to a determination that the aircraft is physically moving toward the capture envelope of the current Flight Plan leg.

hpg-3-Path_Divergence
(broadcast event)

This event corresponds to a determination that the aircraft is physically moving away from the capture envelope of the current Flight Plan leg.

hpg-3-Waypoint_Acquired
(broadcast event)

This event corresponds to a determination that the "current leg" of the active flight plan is complete. This means that the first waypoint in the active flight plan should be deleted, and the aircraft should transition to the next leg.

Display_State
(process)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

Evaluate_Engagement_Criteria
(process)

This process determines whether the HPG can successfully capture the horizontal component of the flight plan.

Evaluate_Disengagement_Criteria
(process)

This process determines whether the HPG must disengage, that is, release control of the aircraft.

Watch_for_Path_Divergence
(process)

This process determines whether the aircraft is progressing towards the first path of the active flight plan, raising a signal if not.

Watch_for_Path_Convergence
(process)

This process determines whether the aircraft is progressing towards the first path of the active flight plan, raising a signal if so.

Fly_Horiz_Comp_of_Active_Flight_Plan

(process)

This process steers the aircraft to, and maintains it on, the horizontal projection of the active flight plan.

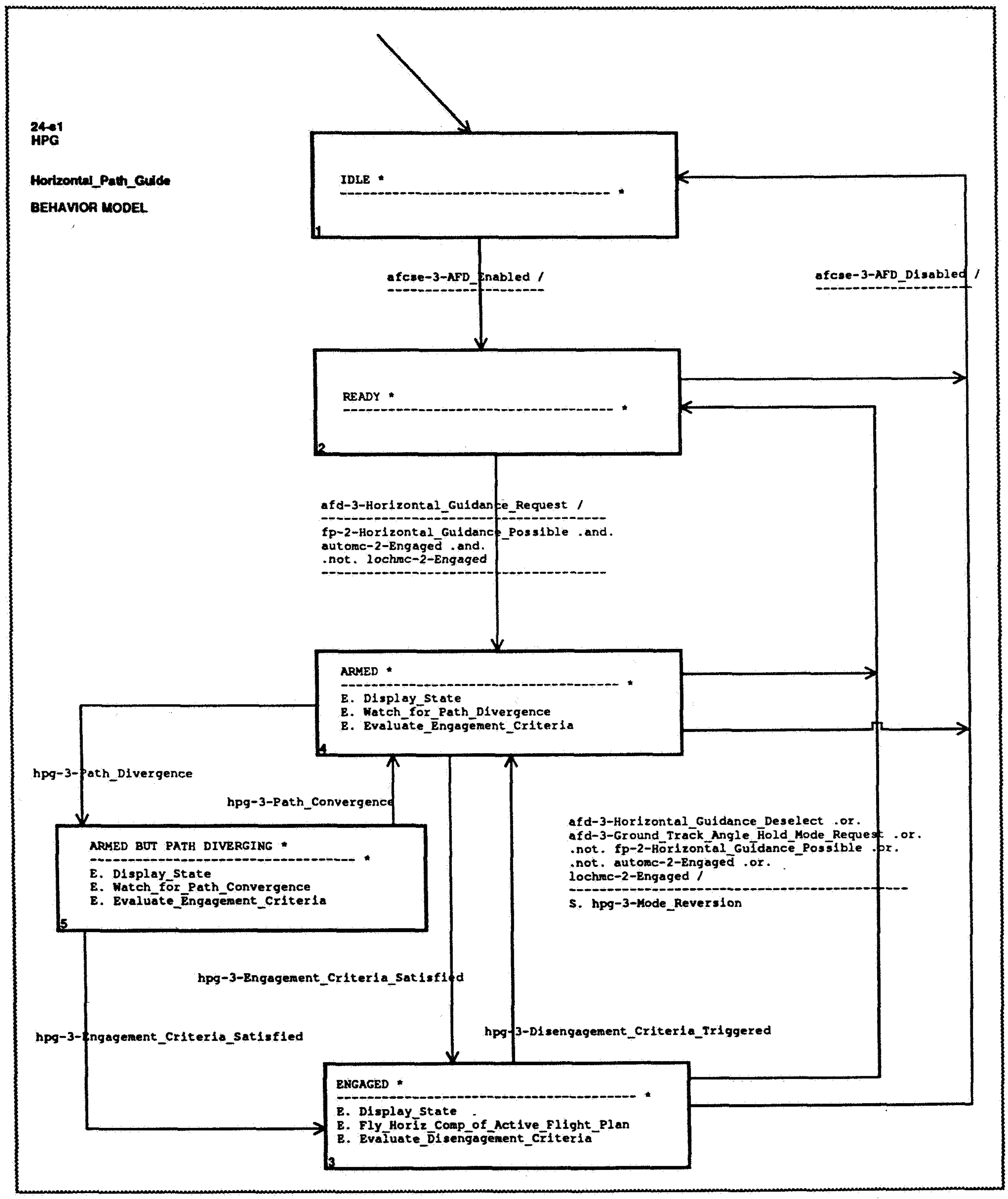


Figure 6.18 Behavior Model Diagram for HPG

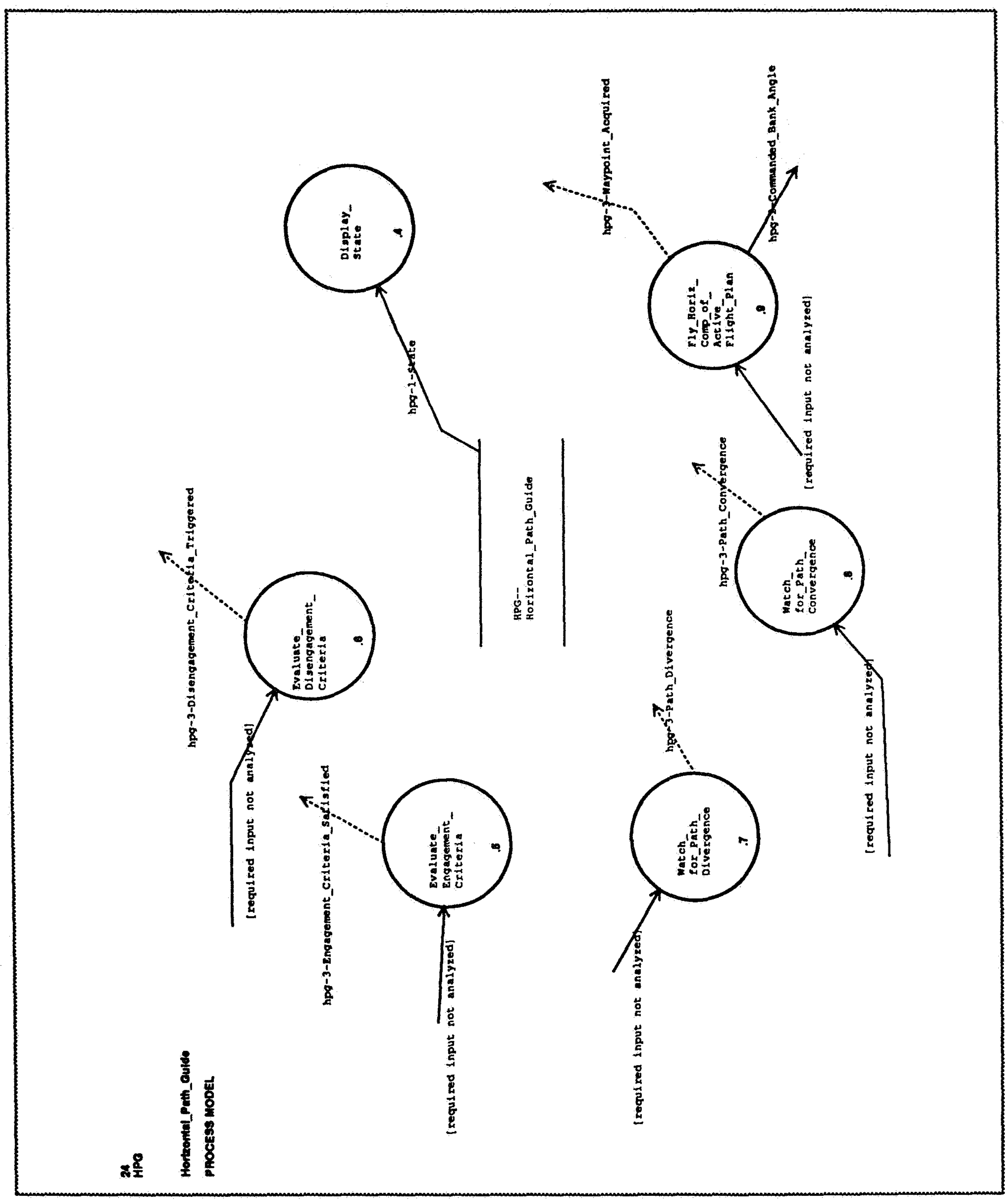


Figure 6.19 Process Model Diagram for HPG

6.1.27 ILSGS--ILS_Ground_System

(*regular object class*) A real-world ILS Ground System is the collection of ground-based signal-generating equipment which forms the ground component of an ILS (Instrument Landing System). The ILSGS object class represents the knowledge within the system that real ILS Ground Systems exist in the real world; members of the ILSGS object class represent the knowledge within the system of particular real-world ILS Ground Systems. In other words, this is NOT an external object class: the system does NOT communicate (directly) with physical, real-world ground-based equipment. (Rather, the system talks to members of the external object class ILSOBS--ILS_On_Board_System, whose radio communication with the real-world ground systems is not documented explicitly.)

A detailed description of an ILS is not given here.

Summary

Primary Identifier:	ilsgs-1-Airport_Name-R15
	+ ilsgs-1-Runway_Name-R15
Attribute:	ilsgs-1-Airport_Name-R15
Attribute:	ilsgs-1-Broadcast_Frequency
Attribute:	ilsgs-1-Glide_Slope_Angle
Attribute:	ilsgs-1-Runway_Name-R15
Attribute:	ilsgs-1-Shack_Position

Definitions

ilsgs-1-Airport_Name-R15
(*referential attribute*)

This is the name of the airport at which this ILS ground system is located.

rw-Airport_Name-R13.

ilsgs-1-Broadcast_Frequency
(*attribute*)

This is the radio frequency used for transmission by the ILS_Ground_-System.

domn-Radio_Frequency.

ilsgs-1-Glide_Slope_Angle
(*attribute*)

This is the designated appropriate approach angle for aircraft landing on the named runway.

Type:	Numeric;
Range:	from 0 to <> degrees;
Accuracy:	to <> degrees;

ilsgs-1-Runway_Name-R15

(referential attribute)

This is the name of the Runway at the named airport which this ILS_-Ground_System services.

rw-Name.

ilsgs-1-Shack_Position

(attribute)

This is the location above the surface of the Earth of the broadcast antenna.

domn-Three_D_Position.

6.1.28 ILSOBS--ILS_On_Board_System

(*external, anonymous object class, no attributes*) The ILS_On_Board_System is an external, anonymous object class which can determine the deviation of the aircraft from the path defined via radio communication by a nearby ILS Ground System.

Summary

Primary Identifier:	*none*
Derived Attribute:	ilsobs-2-Lateral_Deviation
Derived Attribute:	ilsobs-2-Signal_Strength
Derived Attribute:	ilsobs-2-Vertical_Deviation

Definitions

ilsobs-2-Lateral_Deviation

(*derived attribute*)

The lateral component of the aircraft position error.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	<>;

ilsobs-2-Signal_Strength

(*derived attribute*)

A measure of the reliability of the signal being received from the ground-based system by the on-board system.

Type:	Numeric;
Range:	<>;
Accuracy:	<>;

ilsobs-2-Vertical_Deviation

(*derived attribute*)

The vertical component of the aircraft position error.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	<>;

6.1.29 ISN--Intersection

(regular object class) An Intersection is an ATC_Named_Waypoint which is not a Navigational_Aid. Its five-letter name is officially given and its location is known.

Summary

Primary Identifier:	isn-1-Name-R18
Attribute:	isn-1-Name-R18
Attribute:	isn-1-Preferred_Nav_Aid-R19

Definitions

isn-1-Name-R18

(referential attribute)

atcwpt-Name.

isn-1-Preferred_Nav_Aid-R19

(referential attribute)

navaid-Name.

6.1.30 MANEL--Manual_Electric_Mode_Controller

(*anonymous object class, no attributes*) The Manual_Electric_Mode_Controller, while engaged, provides rudimentary transformation of the AFD control inputs; the aircraft is essentially being flown manually.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Derived Attribute:	manel-2-Commanded_Bank_Angle
Derived Attribute:	manel-2-Commanded_Pitch_Angle
Derived Attribute:	manel-2-Commanded_Yaw_Angle
Process:	Enhance_Pilot_Control_Inputs

Definitions

manel-2-Commanded_Bank_Angle
(*derived attribute*)

domn-Bank_Angle.

manel-2-Commanded_Pitch_Angle
(*derived attribute*)

domn-Pitch_Angle.

manel-2-Commanded_Yaw_Angle
(*derived attribute*)

domn-Yaw_Angle.

Enhance_Pilot_Control_Inputs
(*process*)

This process provides rudimentary transformation of the AFD control inputs. The aircraft is essentially being flown manually.

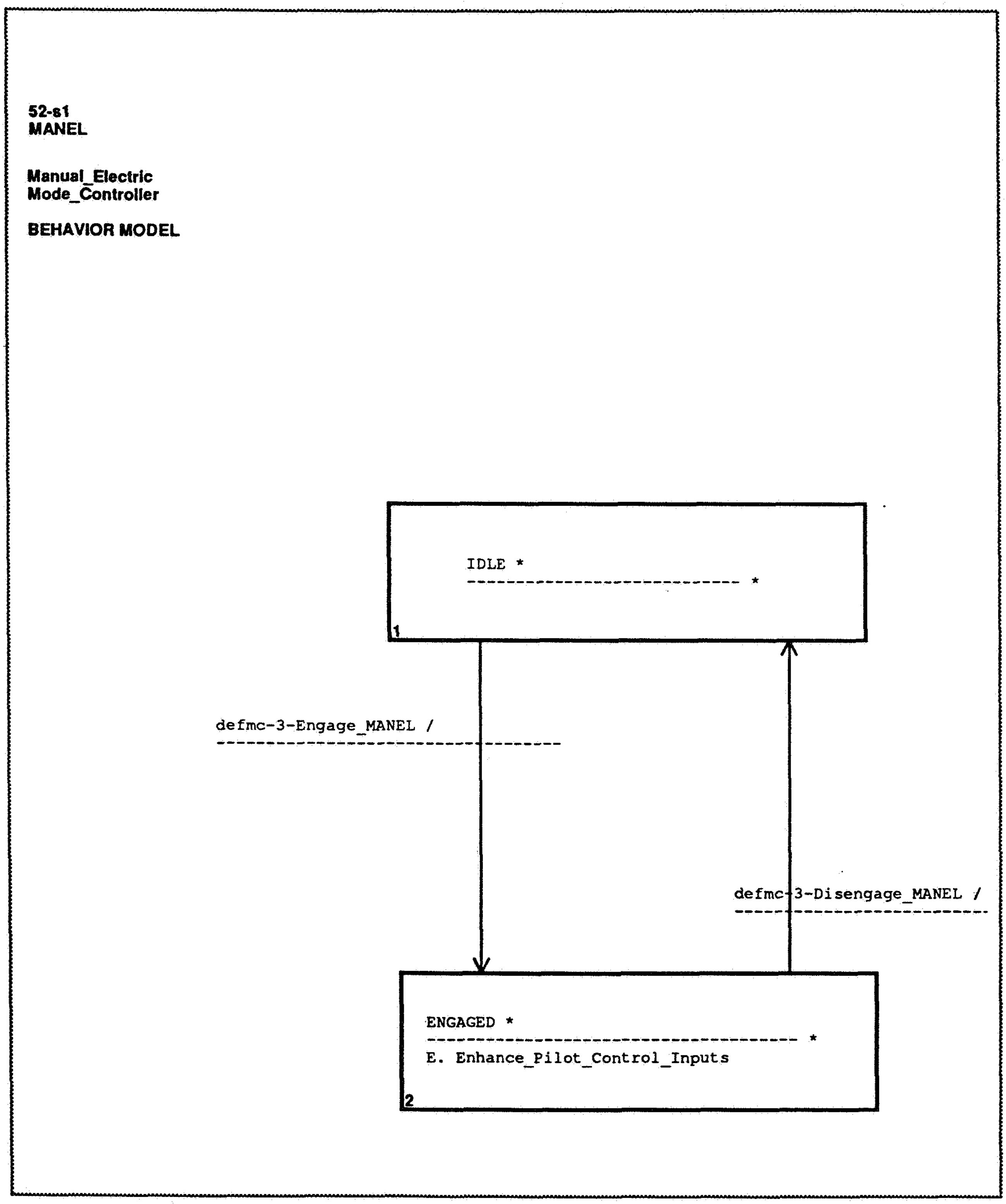


Figure 6.20 Behavior Model Diagram for MANEL

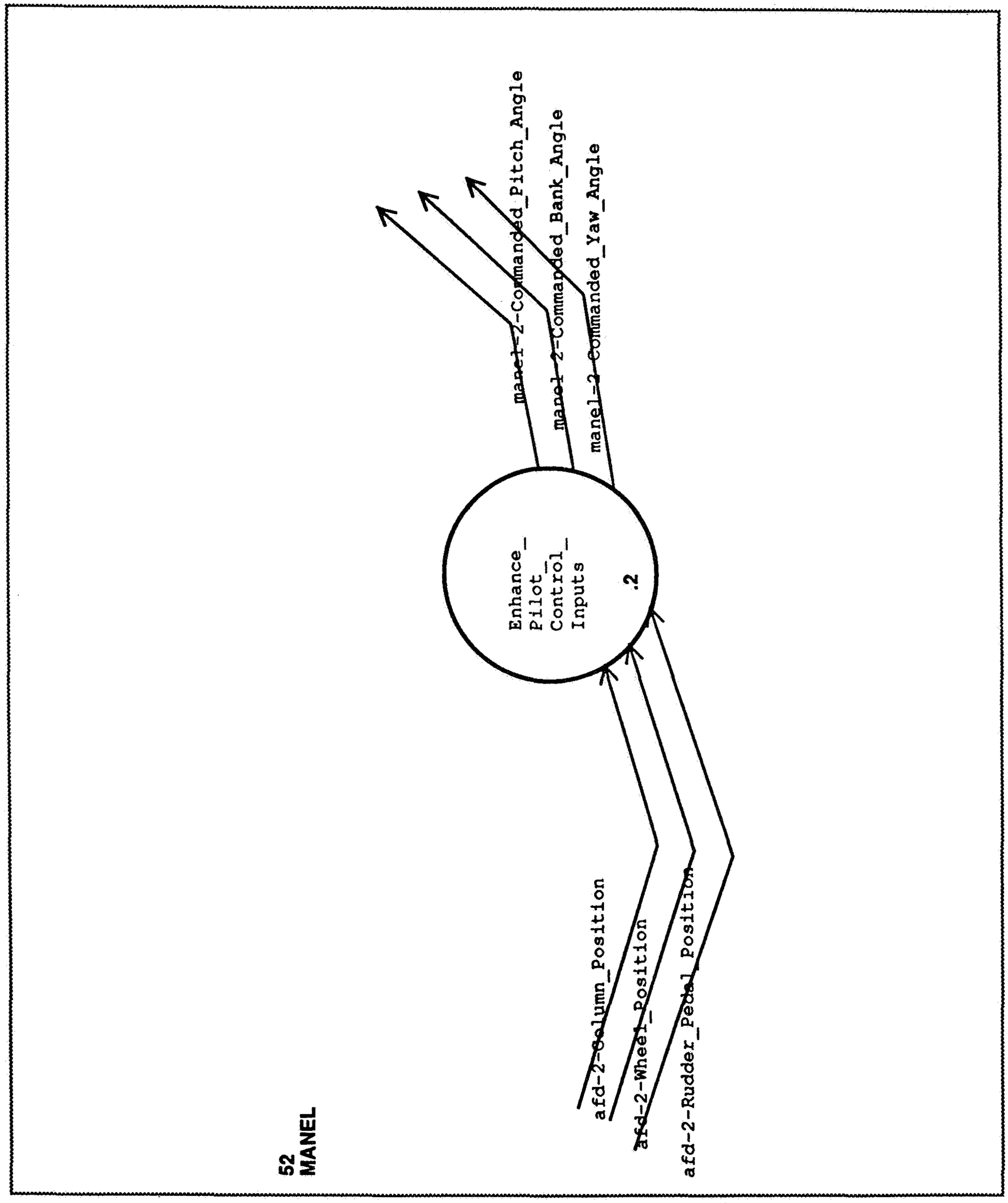


Figure 6.21 Process Model Diagram for MANEL

6.1.31 MLSGS--MLS_Ground_System

(*regular object class*) A real-world MLS Ground System is the collection of ground-based signal-generating equipment which forms the ground component of an MLS (Microwave Landing System). The MLSGS object class represents the knowledge within the system that real MLS Ground Systems exist in the real world. See also ILSGS--ILS_Ground_System.

Summary

Primary Identifier:	mlsgs-1-Airport_Name-R21
	+ mlsgs-1-Runway_Name-R21
Attribute:	mlsgs-1-Airport_Name-R21
Attribute:	mlsgs-1-Azimuth_Broadcast_Location
Attribute:	mlsgs-1-DME_Location
Attribute:	mlsgs-1-Flare_Broadcast_Location
Attribute:	mlsgs-1-Glide_Path_Broadcast_Location
Attribute:	mlsgs-1-Runway_Name-R21

Definitions

mlsgs-1-Airport_Name-R21

(*referential attribute*)

This is the name of the airport at which this MLSGS is located.

rw-Airport_Name-R13.

mlsgs-1-Azimuth_Broadcast_Location

(*compound attribute*)

This is the location of the antenna which broadcasts azimuth information, specified relative to the rw-Threshold of the named runway.

  domn-X_Offset_From_Runway_Threshold
+ domn-Y_Offset_From_Runway_Threshold
+ domn-Z_Offset_From_Runway_Threshold.

mlsgs-1-DME_Location

(*compound attribute*)

This is the location of the antenna which broadcasts distance information, specified relative to the Azimuth_Broadcast_Location.

  domn-X_Offset_From_Azimuth_Broadcast_-Location
+ domn-Y_Offset_From_Azimuth_Broadcast_-Location
+ domn-Z_Offset_From_Azimuth_Broadcast_-Location.

mlsgs-1-Flare_Broadcast_Location
(*compound attribute*)

This is the location of the antenna which broadcasts flare information, specified relative to the Azimuth_Broadcast_Location.

	domn-X_Offset_From_Azimuth_Broadcast_-Location
+	domn-Y_Offset_From_Azimuth_Broadcast_-Location
+	domn-Z_Offset_From_Azimuth_Broadcast_-Location.

mlsgs-1-Glide_Path_Broadcast_Location
(*compound attribute*)

This is the location of the antenna which broadcasts glide slope information, specified relative to the Azimuth_Broadcast_Location.

	domn-X_Offset_From_Azimuth_Broadcast_-Location
+	domn-Y_Offset_From_Azimuth_Broadcast_-Location
+	domn-Z_Offset_From_Azimuth_Broadcast_-Location.

mlsgs-1-Runway_Name-R21
(*referential attribute*)

This is the name of the runway which this MLSGS services.

rw-Name.

6.1.32 MLSOBS--MLS_On_Board_System

(*external, anonymous object class, no attributes*) The MLS_Aircraft_System is an external, anonymous object class which can determine the deviation of the aircraft from the path defined via radio communication by a nearby MLS Ground System.

Summary

Primary Identifier:	*none*
Derived Attribute:	mlsobs-2-Lateral_Deviation
Derived Attribute:	mlsobs-2-Signal_Strength
Derived Attribute:	mlsobs-2-Vertical_Deviation

Definitions

mlsobs-2-Lateral_Deviation
(*derived attribute*)

The lateral component of the aircraft position error.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	<>;

mlsobs-2-Signal_Strength
(*derived attribute*)

A measure of the reliability of the signal being received from the ground-based system by the on-board system.

Type:	Numeric;
Range:	<>;
Accuracy:	<>;

mlsobs-2-Vertical_Deviation
(*derived attribute*)

The vertical component of the aircraft position error.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	<>;

6.1.33 NAVAID--Navigational_Aid

(*regular object class*) A real-world Navigational Aid (aka Nav Aid) is a radio broadcast device located on the surface of the earth designed to enable an aircraft to determine its latitude and longitude (so long as the aircraft is within the service range of the Nav_Aid). The NAVAID object class represents the knowledge within the system that real Navigational Aids exist in the real world; members of the NAVAID object class represent knowledge within the system of particular real-world nav aids. In other words, this is NOT an external object class: the system does NOT communicate (directly) with physical, real-world, on-the-ground nav aids. (Rather, the system communicates only directly with aircraft on-board receivers and other sensors.)

Every NAVAID is an example of an ATCWPT; that is, the NAVAID object class is subtype of the ATCWPT object class.

The navaid-Name is the official name designated by the FAA. Nav_Aids are called out on air navigation charts. "Navigational Aid" is a generally recognized term of the aviation community.

Summary

Primary Identifier:	navaid-1-Name
Alias:	Nav_Aid, nav aid, navaid;
Attribute:	navaid-1-Elevation
Attribute:	navaid-1-Frequency
Attribute:	navaid-1-Name
Attribute:	navaid-1-Service_Range
Attribute:	navaid-1-Type

Definitions

navaid-1-Elevation

(*attribute*)

This is the height of the nav aid's broadcast antenna.

domn-Altitude_Above_Mean_Sea_Level.

navaid-1-Frequency

(*attribute*)

The broadcast frequency of this nav aid. For VOR/DMEs, this is the VOR frequency; DME frequency is derivable from this.

Type:	Numeric;
Range:	from 108.0 to 117.9 MHz;
Accuracy:	to 0.1 MHz;

navaid-1-Name

(*attribute*)

This is the official identifier of the nav aid, as assigned by the FAA.

Type:	String;
Range:	exactly 3 upper case letters;

navaid-1-Service_Range

(attribute)

This indicates the general shape of the airspace above the nav aid for which the nav aid is certified to provide reliable broadcast signals; additional restrictions on this volume, due to mountains and so on, are not recorded within the system.

Type:	Enumeration;
Values:	Low_Altitude, High_Altitude;

navaid-1-Type

(attribute)

This indicates the class of the nav aid.

Type:	Enumeration;
Values:	VOR, VOR-DME, TACAN, NDB, Localizer_Beacon;

6.1.34 PAR--Published_Airway_Route

(*regular object class*) A Published_Airway_Route is an officially recognized sequence of ATCWPTs which form a two-dimensional, bi-directional path across the surface of the earth. The par-1-Number is the official designation of the FAA. Given just any sequence of ATCWPTs, these may or may not form an official PAR. Published_Airway_Routes are called out on air navigation charts.

At the time of this writing, Published_Airway_Routes are divided into two categories (types): "low altitude routes" and "high altitude routes", also known as "Victor Airways" and "Jet Airways" (respectively).

Summary

Primary Identifier:	par-1-Type + par-1-Number.
Attribute:	par-1-Number
Attribute:	par-1-Type

Definitions

par-1-Number
(*attribute*)

This is the unique id of the route, as assigned by the FAA. It is only unique within it's class (type).

Type:	Integer;
Range:	from 1;

par-1-Type
(*attribute*)

This is the class of the route.

Type:	Enumeration;
Values:	Victor, Jet;

6.1.35 PCWPT--Pilot_or_Company_Defined_Waypoint

(*regular object class*) A PCWPT is an unofficial Waypoint, defined by a company as a standard reference for all pilots within the company, or defined by a pilot for use during a particular flight.

Summary

Primary Identifier:	pcwpt-1-Name-R17
Attribute:	pcwpt-1-Name-R17
Attribute:	pcwpt-1-Preferred_Nav_Aid-R05
Attribute:	pcwpt-1-Primary_Use_Description

Definitions

pcwpt-1-Name-R17

(*referential attribute*)

> wpt-1-Name.

pcwpt-1-Preferred_Nav_Aid-R05

(*referential attribute*)

> navaid-1-Name.

pcwpt-1-Primary_Use_Description

(*attribute*)

A short English description of the primary purpose for which this waypoint was created (e.g., "DME arc center").

> Type: String;

6.1.36 RA--Restricted_Area

(regular object class) A Restricted_Area is an air traffic control area requiring explicit entry authorization by the controlling ATC facility. In this analysis, a Restricted_Area always takes the shape of a polygon, which is defined by its vertices (see Restricted_Area_Vertex). "Restricted Area" is an official term; however, we use the term colloquially, so that ADIZs, CADIZs, etc., are included as members of this class.

Summary

Primary Identifier:	ra-1-Name
Attribute:	ra-1-Name
Attribute:	ra-1-Type

Definitions

ra-1-Name

(attribute)

The official name of the restricted area.

Type:	String;

ra-1-Type

(attribute)

Type:	Enumeration;
Values:	Air Defense Intercept Zone, Coastal Air Defense Intercept Zone, Restricted Area, <>;

6.1.37 RAV--Restricted_Area_Vertex

(*regular object class*) A Restricted_Area_Vertex is a vertex of a polygon which defines a Restricted_Area. It does not necessarily coincide with any other geographic reference point. "Restricted Area Vertex" is not a generally used term, but is more or less self-explanatory.

Summary

Primary Identifier:	rav-1-Restricted_Area_Name-R16A + rav-1-Vertex_Number
Attribute:	rav-1-Latitude
Attribute:	rav-1-Longitude
Attribute:	rav-1-Restricted_Area_Name-R16A
Attribute:	rav-1-Vertex_Number

Definitions

rav-1-Latitude

(*attribute*)

The latitude of the vertex.

> domn-Latitude.

rav-1-Longitude

(*attribute*)

The longitude of the vertex.

> domn-Longitude.

rav-1-Restricted_Area_Name-R16A

(*referential attribute*)

The id of the Restricted Area of which this is a vertex.

> ra-Name.

rav-1-Vertex_Number

(*attribute*)

The position of this vertex in the sequential vertex list. The line connecting this vertex with its successor defines one edge of the polygonal Restricted Area. Numbering the vertices is necessary to unambiguously define the polygon.

Type:	Integer
Range:	from 1;

6.1.38 RUD--Rudder

(*external, anonymous object class, no attributes*) The Rudder is one of the aerodynamic control effectors used to control the motion of the aircraft.

Summary

Primary Identifier:	*none*
Derived Attribute:	rud-2-Current_Position

Definitions

rud-2-Current_Position
(*derived attribute*)

The current sensed position of the rudder.

domn-Rudder_Position.

6.1.39 RW--Runway

(*regular object class*) A Runway is a strip of pavement on which an aircraft lands in a particular direction. A given strip of pavement (airstrip) plays the role, at different times, of two distinct Runways (e.g., 10L and 28R). Thus the term "Runway" refers to one of the roles played by an airstrip rather than to the physical airstrip itself.

Summary

Primary Identifier:	rw-1-Name
	+ rw-1-Airport_Name-R13
Attribute:	rw-1-Airport_Name-R13
Attribute:	rw-1-Magnetic_Heading
Attribute:	rw-1-Missed_Appr_Point
Attribute:	rw-1-Name
Attribute:	rw-1-Threshold
Attribute:	rw-1-Usable_Length

Definitions

rw-1-Airport_Name-R13
(*referential attribute*)

The name of the Airport at which this Runway is located.

ap-Name.

rw-1-Magnetic_Heading
(*attribute*)

The magnetic heading of the Runway.

Type:	Numeric;
Range:	from 0 to 360 degrees;
Accuracy:	to 1 degree;

rw-1-Missed_Appr_Point
(*attribute*)

The location on the surface of the earth over which the pilot must make his go-around decision.

domn-Two_D_Position.

rw-1-Name

(attribute)

This is the official name of the Runway, which is determined by (1) its Magnetic_Heading, and (2) its position at the Airport relative to a runway parallel to it (if one exists). In the format "nh", 'n' is a string of 1 to 2 digits equal to the magnetic heading rounded to the nearest 10 degrees; 'h' is either "L" or "R" (for left-hand runway and right-hand runway, respectively). If there is no parallel runway, then the 'h' is dropped.

Type:	String;
Range:	from 1 to 3 characters;
Format:	nh;

rw-1-Threshold

(attribute)

The location of the beginning of the usable part of the Runway.

domn-Two_D_Position.

rw-1-Usable_Length

(attribute)

The usable length of the Runway, starting from the Threshold.

Type:	Numeric;
Range:	from 0 to <> feet;
Accuracy:	to 1 foot;

6.1.40 SDF--Sensor_Data_Filterer

(*external, anonymous object class, no attributes*) The Sensor_Data_Filterer reads the raw data provided by the external object class SENS-Sensor_Suite, translating, filtering and validating this data. Some of the data provided by the SENS-Sensor_Suite may be simply "passed through". The mapping from sensed data to SDF data is non-trivial.

Note: this object class is a "pseudo external object class": treated as external until such time as this aspect of the system can be analyzed further. For the interim, this object class provides a source and sink for closely related events and data. Further analysis may reveal that this single placeholder must become several object classes. Also, several of these pseudo external object classes might be split and joined. The array of inputs from the sensor suite is not known at the time of this writing.

Summary

Primary Identifier:	*none*
Derived Attribute:	sdf-2-Altitude_Above_Ground_Level
Derived Attribute:	sdf-2-Altitude_Above_Mean_Sea_Level
Derived Attribute:	sdf-2-Angle_of_Attack
Derived Attribute:	sdf-2-Bank_Angle
Derived Attribute:	sdf-2-Bank_Angle_Rate_of_Change
Derived Attribute:	sdf-2-Flight_Path_Angle
Derived Attribute:	sdf-2-Ground_Track_Angle
Derived Attribute:	sdf-2-Latitude
Derived Attribute:	sdf-2-Longitude
Derived Attribute:	sdf-2-Pitch_Angle
Derived Attribute:	sdf-2-Pitch_Angle_Rate_of_Change
Derived Attribute:	sdf-2-Sideslip_Angle
Derived Attribute:	sdf-2-True_Airspeed
Derived Attribute:	sdf-2-Wind_Velocity
Derived Attribute:	sdf-2-X_Acceleration
Derived Attribute:	sdf-2-Y_Acceleration
Derived Attribute:	sdf-2-Yaw_Angle
Derived Attribute:	sdf-2-Yaw_Angle_Rate_of_Change
Derived Attribute:	sdf-2-Z_Acceleration

Definitions

sdf-2-Altitude_Above_Ground_Level
(*derived attribute*)

This is the current altitude of the aircraft.

domn-Altitude_Above_Ground_Level.

sdf-2-Altitude_Above_Mean_Sea_Level
(*derived attribute*)

This is the current altitude of the aircraft.

domn-Altitude_Above_Mean_Sea_Level.

sdf-2-Angle_of_Attack
(derived attribute)

This is the angle of attack of the aircraft.

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

sdf-2-Bank_Angle
(derived attribute)

domn-Bank_Angle.

sdf-2-Bank_Angle_Rate_of_Change
(derived attribute)

Type:	Numeric;
Range:	from <> to <> degrees per second;
Accuracy:	<>;

sdf-2-Flight_Path_Angle
(derived attribute)

domn-Flight_Path_Angle.

sdf-2-Ground_Track_Angle
(derived attribute)

domn-Ground_Track_Angle.

sdf-2-Latitude
(derived attribute)

This is the current latitude of the aircraft.

domn-Latitude.

sdf-2-Longitude
(derived attribute)

This is the current longitude of the aircraft.

domn-Longitude.

sdf-2-Pitch_Angle
(derived attribute)

domn-Pitch_Angle.

sdf-2-Pitch_Angle_Rate_of_Change
(derived attribute)

Type:	Numeric;
Range:	from <> to <> degrees per second;
Accuracy:	<>;

sdf-2-Sideslip_Angle

(derived attribute)

This is the sideslip angle of the aircraft.

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

sdf-2-True_Airspeed

(derived attribute)

This is the true airspeed of the aircraft.

domn-Airspeed.

sdf-2-Wind_Velocity

(derived attribute)

Type:	Numeric;
Range:	from <> to <> knots;
Accuracy:	<>;

sdf-2-X_Acceleration

(derived attribute)

This is the instantaneous longitudinal acceleration of the aircraft, where forward acceleration is positive.

domn-Acceleration.

sdf-2-Y_Acceleration

(derived attribute)

This is the instantaneous lateral acceleration of the aircraft, where acceleration towards the right wing is positive.

domn-Acceleration.

sdf-2-Yaw_Angle

(derived attribute)

domn-Yaw_Angle.

sdf-2-Yaw_Angle_Rate_of_Change

(derived attribute)

Type:	Numeric;
Range:	from <> to <> degrees per second;
Accuracy:	<>;

sdf-2-Z_Acceleration

(derived attribute)

This is the instantaneous "vertical" acceleration of the aircraft, where the vertical axis completes the right-hand coordinate system of the aircraft.

domn-Acceleration.

6.1.41 SENS--Sensor_Suite

(external, anonymous object class, no attributes) The Sensor_Suite is a placeholder, an external object class used to provide a simplified source of sensed data until such time as this aspect of the system can be analyzed further. The array of data provided by the Sensor_Suite (to the SDF) is not known; hence there are no definitions given for this object class of any derived attributes. Rather, this object class serves merely to document the existence of some set of raw data, which can be used to derive the set of data provided by the SDF.

Note that external object classes which have already been modeled provide their own sensed data. The SENS object class exists as a "catch-all" for sensed data which does not have a clear source at the time of this writing.

Summary

Primary Identifier: *none*

6.1.42 SS--SID_or_STAR

(regular object class) A member of this object class is either a SID or a STAR; these two FAA procedures were modeled as a single object class because their data, behavior and process models are identical. This is due to the fact that the system shall give maximum flexibility to the pilot as he or she constructs a Flight_Plan. (That is, the system shall not constrain SIDs to be at the beginning, etc.) Maximum flexibility manifests itself as very generic behavior and processing for SIDs and STARs.

A Standard_Instrument_Departure_Procedure is officially referred to as a SID. A SID is a published IFR ATC departure procedure established by the FAA. SIDs exist to reduce communication in the terminal area of an airport by providing a predefined transition route from a particular Runway to an appropriate en route structure. The ss-1-Name shall be the official, published SID name. "Standard Instrument Departure (SID)" is an official term.

A Standard_Terminal_ARrival_Procedure is officially referred to as a STAR. A STAR is a published IFR ATC arrival procedure established by the FAA. STARs exist to reduce communication in the terminal area of an airport by providing a predefined transition route from an en route structure to a particular Runway. The ss-1-Name shall be the official, published STAR name. "Standard Terminal ARrival (STAR)" is an official term.

Note that combining SIDs and STARs into a single SS object class did introduce a couple "wrinkles": the introduction of the "Type" attribute, and the transitive dependence of "IAS Required" on the key. However, the savings in terms of reduced complexity provided the impetus.

Summary

Primary Identifier:	ss-1-Name + ss-1-Type
Attribute:	ss-1-Airport_Name-R12B-R14B-R20B
Attribute:	ss-1-IAS_Required-R14B
Attribute:	ss-1-Name
Attribute:	ss-1-Runway_Name-R12B-R14B-R20B
Attribute:	ss-1-Type

Definitions

ss-1-Airport_Name-R12B-R14B-R20B

(referential attribute)

The id of the Airport at which this SS is defined.

rw-1-Airport_Name-R13.

ss-1-IAS_Required-R14B

(subtype indication)

The type of Instrument Approach System (IAS) defined for this procedure. Must be "None" for STARs; may be "None" for SIDs.

Type:	Enumeration;
Values:	ILS, MLS, None;

ss-1-Name
(attribute)

The official id of the SID/STAR, unique across all Airports.

Type:	String;
Range:	<>;
Format:	<>;

ss-1-Runway_Name-R12B-R14B-R20B
(referential attribute)

The id of the Runway for which this procedure is defined; for SIDs, this is the Runway of origin; for STARs, this is the destination Runway.

rw-1-Name.

ss-1-Type
(attribute)

The class of procedure.

Type:	Enumeration;
Values:	SID, STAR;

6.1.43 STAB--Stabilizer

(*external, anonymous object class, no attributes*) The Stabilizer is one of the aerodynamic control effectors used to control the motion of the aircraft.

Summary

Primary Identifier:	*none*
Derived Attribute:	stab-2-Current_Position

Definitions

stab-2-Current_Position
(*derived attribute*)

The current sensed position of the Stabilizer.

domn-Stabilizer_Position.

6.1.44 TOBJ--Terrain_Object

(*regular object class*) A Terrain Object is any physical obstruction to flight of which a pilot should be aware. A Terrain Object known to the system can appear on a map display, for instance, or be used to validate a flight plan.

Summary

Primary Identifier:	tobj-1-Latitude
	+ tobj-1-Longitude
Attribute:	tobj-1-Elevation
Attribute:	tobj-1-Latitude
Attribute:	tobj-1-Longitude
Attribute:	tobj-1-Type

Definitions

tobj-1-Elevation
(*attribute*)

domn-Altitude_Above_Ground_Level.

tobj-1-Latitude
(*attribute*)

domn-Latitude.

tobj-1-Longitude
(*attribute*)

domn-Longitude.

tobj-1-Type
(*attribute*)

Type:	Enumeration;
Values:	Mountain, Obstruction;

6.1.45 TPG--Time_Path_Guide

(*anonymous object class, no attributes*) The TPG, while engaged, generates steering commands that direct the aircraft to, and maintain it on, the time projection of the Flight_Plan. The TPG assumes the aircraft is near the leg between the first two waypoints of the flight plan.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Derived Attribute:	tpg-2-Commanded_Airspeed
Process:	Display_State
Process:	Fly_Time_Comp_of_Active_Flight_Plan

Definitions

tpg-2-Commanded_Airspeed
(*derived attribute*)

This is the commanded airspeed which the aircraft shall attain.

$\boxed{\text{domn-Airspeed.}}$

Display_State
(*process*)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

Fly_Time_Comp_of_Active_Flight_Plan
(*process*)

This process steers the aircraft to, and maintains it on, the time projection of the active flight plan.

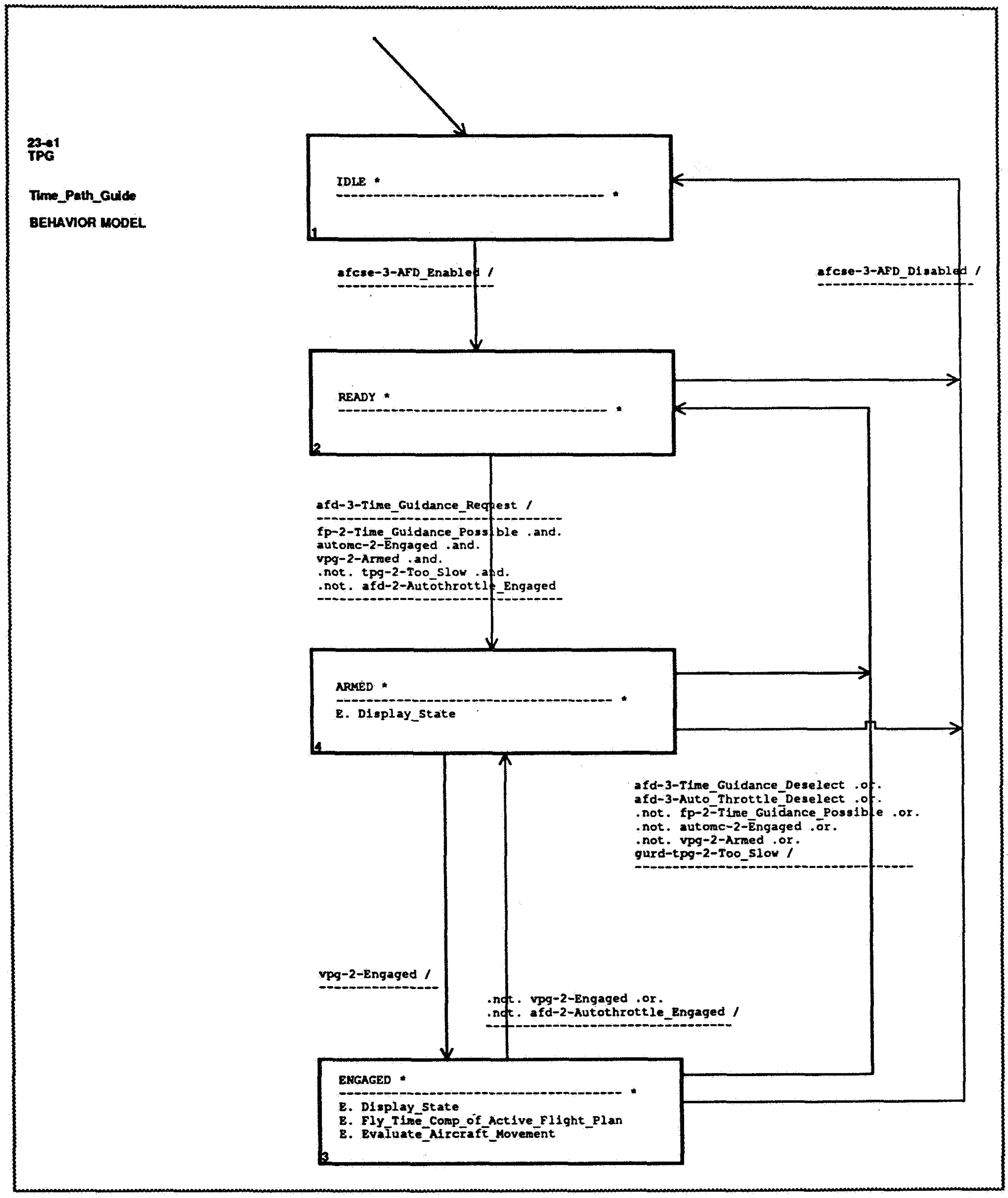


Figure 6.22 Behavior Model Diagram for TPG

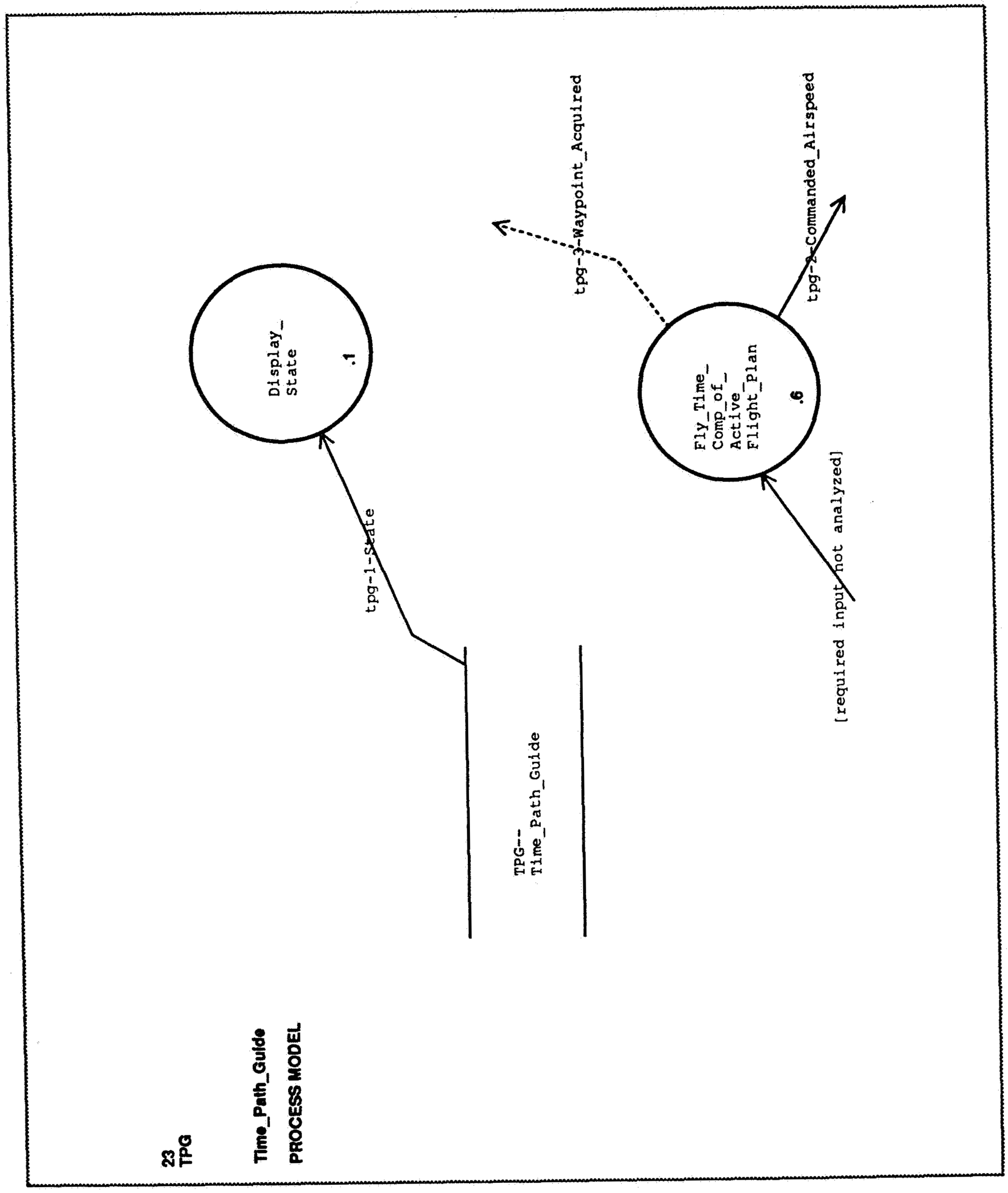


Figure 6.23 Process Model Diagram for TPG

6.1.46 VCWS--Velocity_Vector_Hold_CWS_Mode_Controller

(*anonymous object class*) The VCWS Mode Controller, while engaged, maintains the velocity vector (i.e., ground track angle and flight path angle) of the aircraft by issuing commands to the Effector_Controller. The VCWS allows the AFD_Crew to change the velocity-vector-being-maintained by responding to wheel/column inputs (or equivalent control inputs) while engaged.

Summary

Primary Identifier:	*none*
Alias:	VCWS Mode Controller
Behavior Model:	*present*
Process Model:	*present*
Attribute:	vcws-1-Selected_Flight_Path_Angle
Attribute:	vcws-1-Selected_Ground_Track_Angle
Derived Attribute:	vcws-2-Commanded_Bank_Angle
Derived Attribute:	vcws-2-Commanded_Vertical_Acceleration
Derived Attribute:	vcws-2-Velocity_Vector_Hold_Engaged
Process:	Set_Selected_Velocity_Vector_to_Actual
Process:	Maintain_Velocity_Vector_of_Aircraft

Definitions

vcws-1-Selected_Flight_Path_Angle
(*attribute*)

domn-Flight_Path_Angle.

vcws-1-Selected_Ground_Track_Angle
(*attribute*)

domn-Ground_Track_Angle.

vcws-2-Commanded_Bank_Angle
(*derived attribute*)

domn-Bank_Angle.

vcws-2-Commanded_Vertical_Acceleration
(*derived attribute*)

domn-Acceleration.

vcws-2-Velocity_Vector_Hold_Engaged
(*derived attribute*)

"True" when the VCWS is engaged; "False" otherwise.

domn-Boolean.

Set_Selected_Velocity_Vector_to_Actual

(process)

This process makes the selected velocity vector (flight path angle and ground track angle) of the aircraft equal to the instantaneous actual velocity vector.

Maintain_Velocity_Vector_of_Aircraft

(process)

This process commands the Effector_Controller to maintain the selected velocity vector (flight path angle and ground track angle) of the aircraft, while at the same time allowing the Aft Flight Deck Crew to modify this selected velocity vector via control inputs.

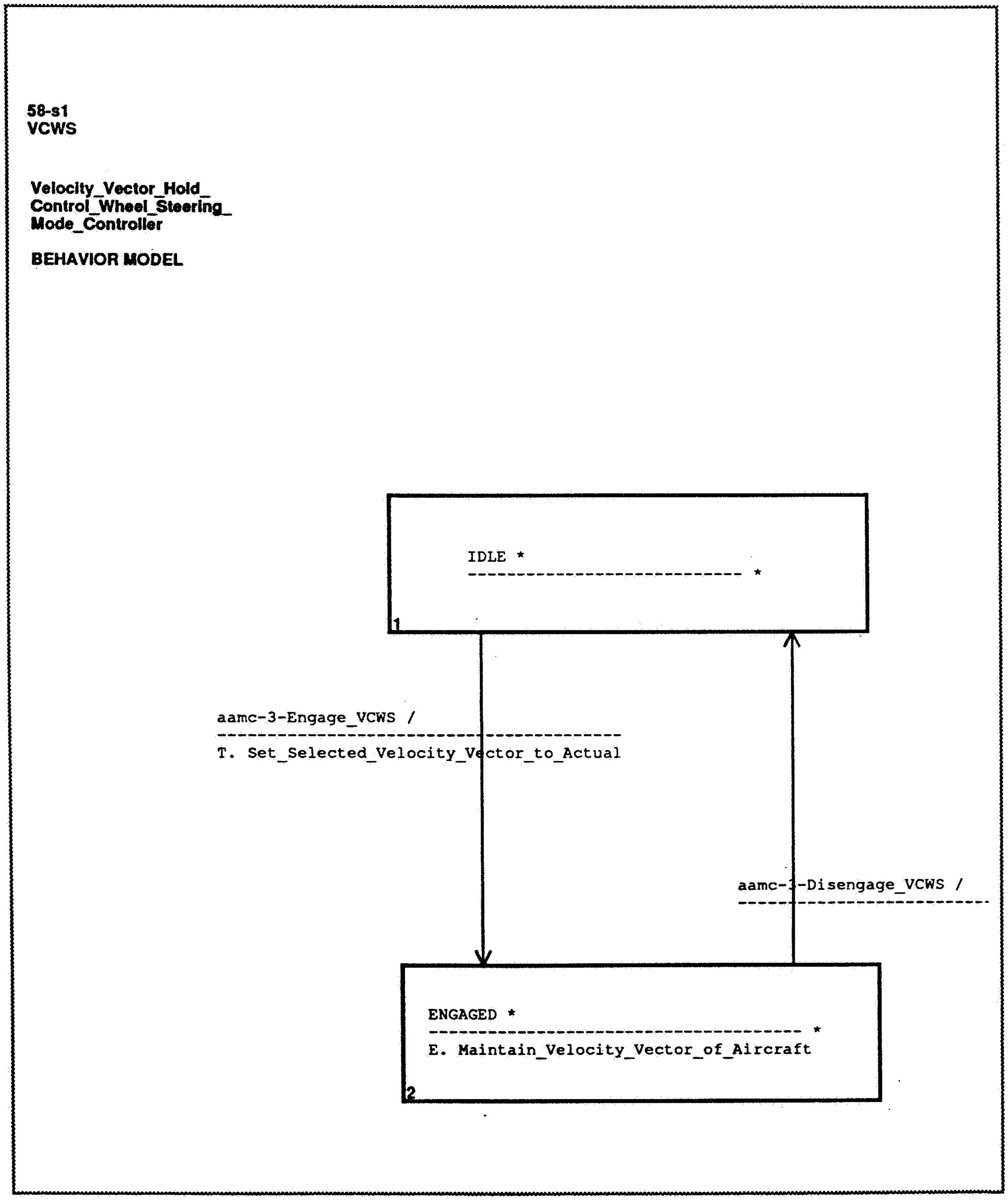


Figure 6.24 Behavior Model Diagram for VCWS

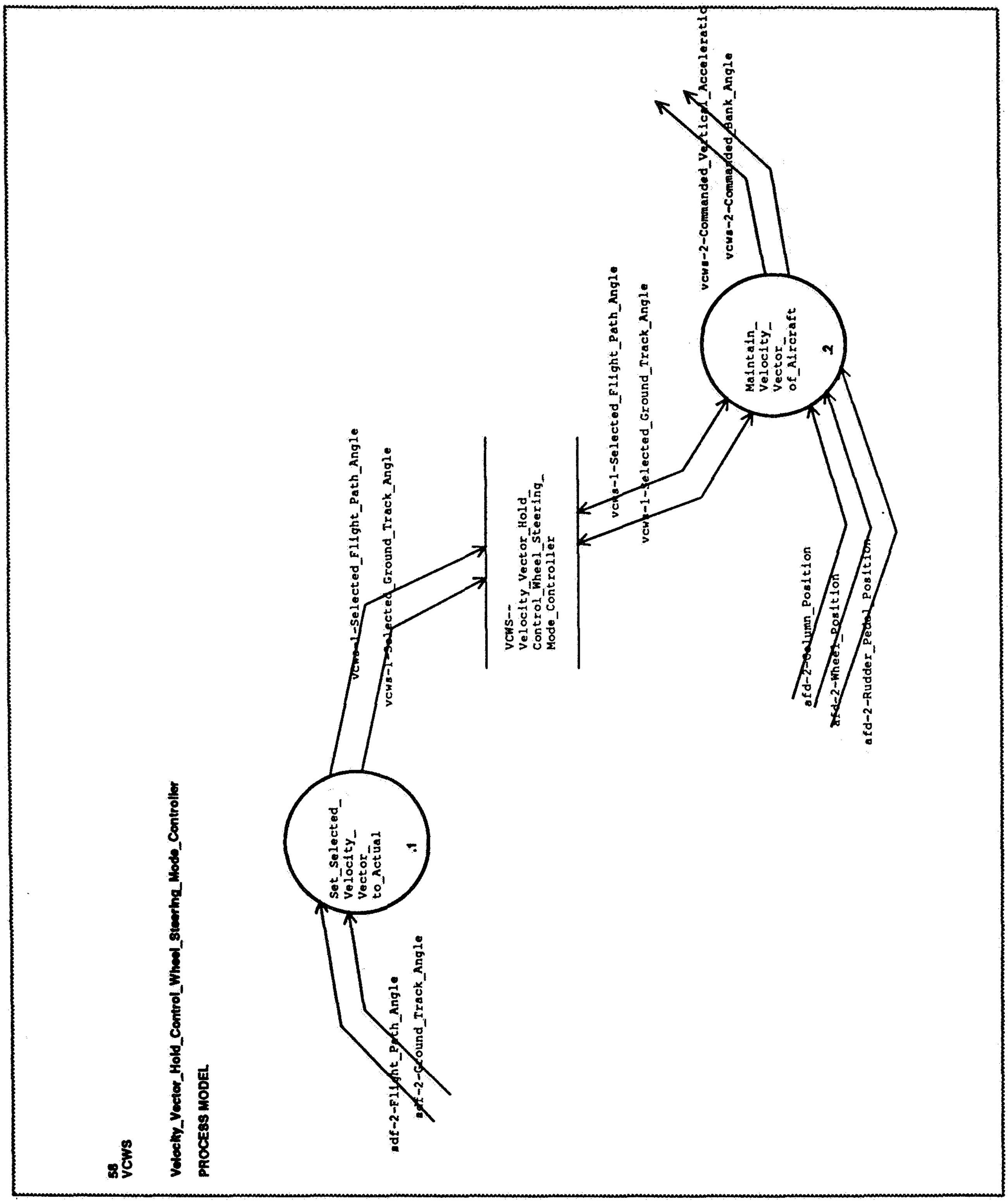


Figure 6.25 Process Model Diagram for VCWS

6.1.47 VPG--Vertical_Path_Guide

(*anonymous object class, no attributes*) The VPG, while engaged, generates steering commands that direct the aircraft to, and maintain it on, the vertical projection of the Flight_Plan. The VPG assumes the aircraft is near the leg between the first two waypoints of the flight plan.

Summary

Primary Identifier:	*none*
Behavior Model:	*present*
Process Model:	*present*
Derived Attribute:	vpg-2-Armed
Derived Attribute:	vpg-2-Commanded_Vertical_Acceleration
Derived Attribute:	vpg-2-Engaged
Event:	vpg-3-Engagement_Criteria_Satisfied
Event:	vpg-3-Mode_Reversion
Event:	vpg-3-Path_Convergence
Event:	vpg-3-Path_Divergence
Process:	Display_State
Process:	Evaluate_Engagement_Criteria
Process:	Evaluate_Disengagement_Criteria
Process:	Watch_for_Path_Divergence
Process:	Watch_for_Path_Convergence
Process:	Fly_Vertical_Comp_of_Active_Flight_Plan

Definitions

vpg-2-Armed
(*derived attribute*)

"True" when the VPG is either armed or engaged; "False" otherwise.

domn-Boolean.

vpg-2-Commanded_Vertical_Acceleration
(*derived attribute*)

This is the commanded vertical acceleration which shall be attained.

domn-Vertical_Acceleration.

vpg-2-Engaged
(*derived attribute*)

"True" when the VPG is engaged; "False" otherwise.

domn-Boolean.

vpg-3-Engagement_Criteria_Satisfied
(*broadcast event*)

This event corresponds to a determination that the VPG can successfully capture the vertical component of the flight plan.

vpg-3-Mode_Reversion
(broadcast event)

This event corresponds to the self-disengagement of the VPG. The event is generated to let others in the system react as they will.

vpg-3-Path_Convergence
(broadcast event)

This event corresponds to a determination that the aircraft is physically moving toward the vertical projection of the flight plan.

vpg-3-Path_Divergence
(broadcast event)

This event corresponds to a determination that the aircraft is physically moving away from the vertical projection of the flight plan.

Display_State
(process)

This process makes visible the current state of this object. (The implementation of this process may display a message on a CRT, or it may light different colored lamps.)

Evaluate_Engagement_Criteria
(process)

This process determines whether the VPG can successfully capture the vertical component of the flight plan.

Evaluate_Disengagement_Criteria
(process)

This process determines whether the VPG must disengage, that is, release control of the aircraft.

Watch_for_Path_Divergence
(process)

This process determines whether the aircraft is progressing towards the first path of the active flight plan, raising a signal if not.

Watch_for_Path_Convergence
(process)

This process determines whether the aircraft is progressing towards the first path of the active flight plan, raising a signal if so.

Fly_Vertical_Comp_of_Active_Flight_Plan
(process)

This process steers the aircraft to, and maintains it on, the vertical projection of the active flight plan.

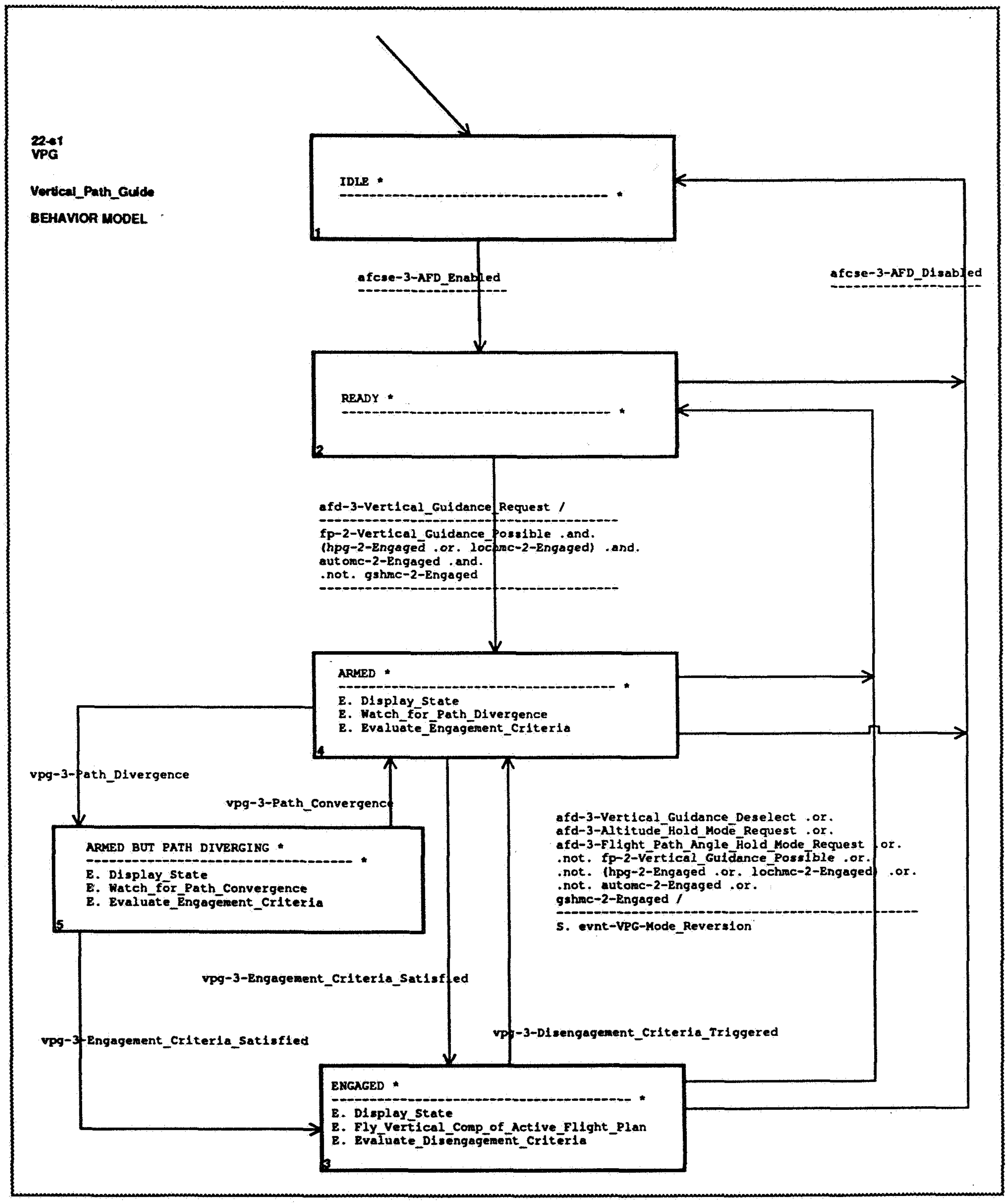


Figure 6.26 Behavior Model Diagram for VPG

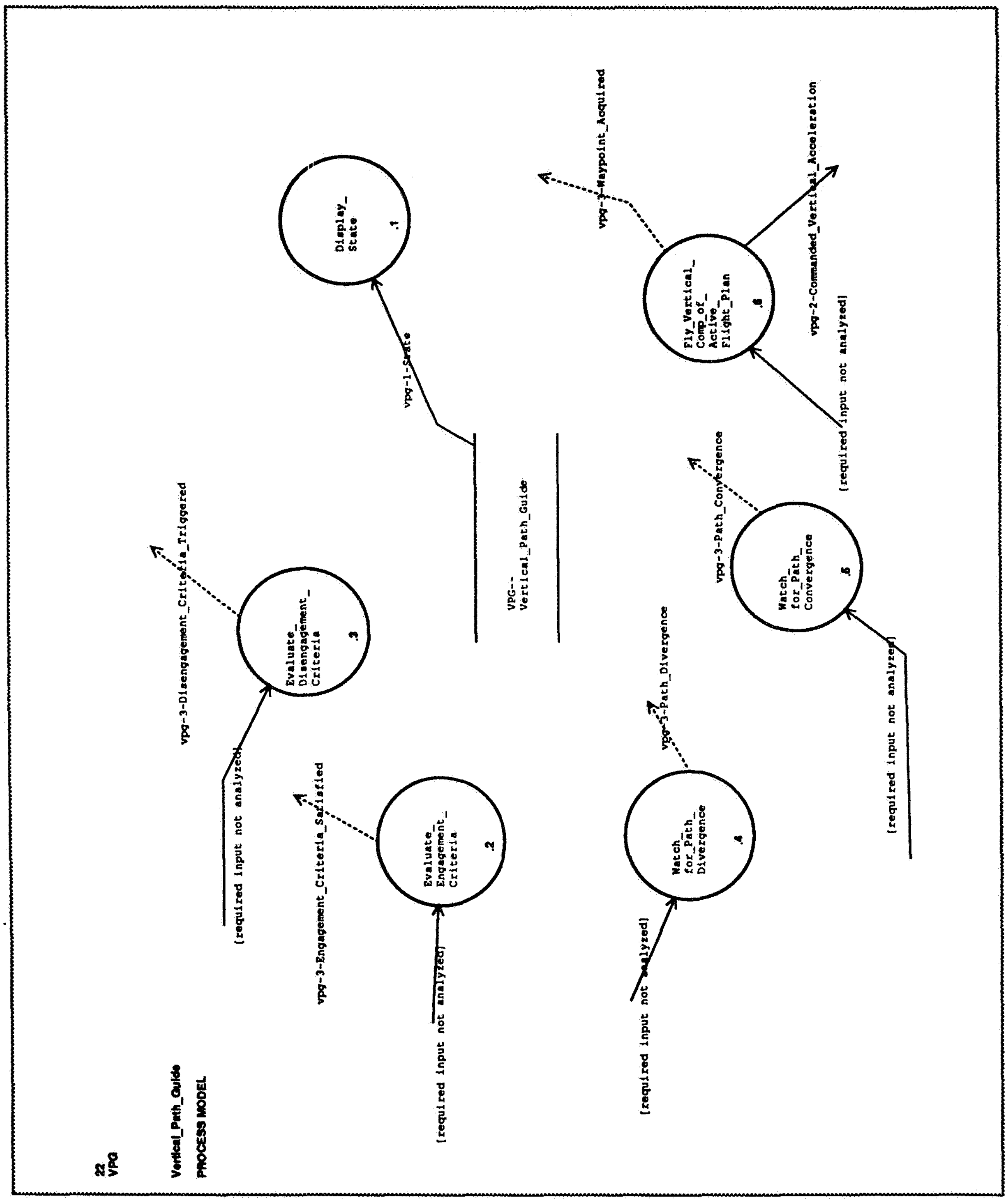


Figure 6.27 Process Model Diagram for VPG

6.1.48 WCR--Waypoint_ona_Company_Route

(*regular object class*) A Waypoint_ona_Company_Route arises out of the association of a particular Waypoint (WPT) with a particular Company Route (CR). It captures the information relevant to such an association. (The same Waypoint on a different Company Route may be assigned different data values.)

"Waypoint on a Company Route" is a somewhat artificial term created for the sake of clarity in this analysis; in general usage, the term "Waypoint" may refer to Waypoints, Waypoints on Company Routes, Waypoints on Flight Plans, or even Waypoints on SIDs or STARs (where the meaning is clear from the context).

Summary

Primary Identifier:	wcr-1-Route_Name-R09B
Attribute:	wcr-1-Route_Name-R09B
Attribute:	wcr-1-Waypoint_Name-R09A

Definitions

wcr-1-Route_Name-R09B

(*referential attribute*)

The id of the Company Route which the named Waypoint is on by means of this WCR.

cr-1-Name.

wcr-1-Waypoint_Name-R09A

(*referential attribute*)

The id of the Waypoint which is on the named Company Route by means of this WCR.

wpt-1-Name.

6.1.49 WFP--Waypoint_ona_Flight_Plan

(*regular object class*) A Waypoint_ona_Flight_Plan arises out of the association of a particular Waypoint with a particular Flight_Plan. It captures the information relevant to such an association. (The same Waypoint on a different Flight_Plan may be assigned different data values.)

"Waypoint on a Flight Plan" is a somewhat artificial term created for the sake of clarity in this analysis; in general usage, the term "Waypoint" may refer to both Waypoints and Waypoints on Flight Plans (where the meaning is clear from the context). For that matter, the term Waypoint may refer to a Waypoint on a SID or STAR (see WSS).

Summary

Primary Identifier:	wfp-1-Flight_Plan_Name-R01A + wfp-1-Waypoint_Name-R01B
Alias:	Waypoint_Enrollment, Enrollment;
Attribute:	wfp-1-Flight_Plan_Name-R01A
Attribute:	wfp-1-Method_of_Acquisition-R03
Attribute:	wfp-1-Ordinal_Position_on_Plan
Attribute:	wfp-1-Type-R02
Attribute:	wfp-1-Waypoint_Name-R01B

Definitions

wfp-1-Flight_Plan_Name-R01A

(*referential attribute*)

The id of the Flight_Plan which the named Waypoint is on by means of this WFP.

fp-1-Name.

wfp-1-Method_of_Acquisition-R03

(*subtype indication*)

The manner in which the aircraft will move from the previous WFP to this WFP.

Type:	Subtype Indication;
Objects:	DA, DMEAA;

wfp-1-Ordinal_Position_on_Plan

(*attribute*)

The sequential position (first, second, third, ...) of this WFP on the named Flight_Plan.

Type:	Integer;
Range:	from 1;

wfp-1-Type-R02

(subtype indication)

The type of WFP this is; this points the way to further information concerning this WFP (if any).

Type:	Subtype Indication;
Objects:	WFP2D, WFP3D, WFP4D, CWFP;

wfp-1-Waypoint_Name-R01B

(referential attribute)

The id of the Waypoint which is on the named Flight_Plan by means of this WFP.

wpt-1-Name.

6.1.50 WFP2D--Two-D_Waypoint_ona_Flight_Plan

(*anonymous object class, no attributes*) A WFP2D is a two-dimensional WFP, i.e., a waypoint on a flight plan which has not been assigned additional constraints, such as an altitude for the aircraft to be at, planned time of arrival, etc. As a vacuous object class, it's existence is justified only to complete the subtypes of WFP: it ensures that every WFP has a well-defined wfp-1-Type.

Summary

Primary Identifier: *none*

6.1.51 WFP3D--Three-D_Waypoint_ona_Flight_Plan

(*regular object class*) A WFP3D is a three-dimensional WFP, i.e., a waypoint on a flight plan which has been augmented with altitude information.

Summary

Primary Identifier:	wfp3d-1-Flight_Plan_Name-R02
	+ wfp3d-1-Waypoint_Name-R02
Attribute:	wfp3d-1-Altitude_to_be_at
Attribute:	wfp3d-1-Flight_Plan_Name-R02
Attribute:	wfp3d-1-Waypoint_Name-R02

Definitions

wfp3d-1-Altitude_to_be_at
(*attribute*)

The desired altitude to be at by the time the airplane arrives at the Waypoint.

domn-Altitude_Above_Mean_Sea_Level.

wfp3d-1-Flight_Plan_Name-R02
(*referential attribute*)

wfp-1-Flight_Plan_Name-R01A.

wfp3d-1-Waypoint_Name-R02
(*referential attribute*)

wfp-1-Waypoint_Name-R01B.

6.1.52 WFP4D--Four-D_Waypoint_ona_Flight_Plan

(*regular object class*) A WFP4D is a four-dimensional WFP, i.e., a waypoint on a flight plan which has been augmented with both altitude and time-of-arrival information.

Summary

Primary Identifier:	wfp4d-1-Flight_Plan_Name-R02
	+ wfp4d-1-Waypoint_Name-R02
Attribute:	wfp4d-1-Altitude_to_be_at
Attribute:	wfp4d-1-Flight_Plan_Name-R02
Attribute:	wfp4d-1-Planned_Time_of_Arrival
Attribute:	wfp4d-1-Waypoint_Name-R02

Definitions

wfp4d-1-Altitude_to_be_at
(*attribute*)

wfp3d-Altitude_to_be_at.

wfp4d-1-Flight_Plan_Name-R02
(*referential attribute*)

wfp-1-Flight_Plan_Name-R01A.

wfp4d-1-Planned_Time_of_Arrival
(*attribute*)

The time of day at which the airplane should arrive at the waypoint.

domn-Time_of_Day.

wfp4d-1-Waypoint_Name-R02
(*referential attribute*)

wfp-1-Waypoint_Name-R01B.

6.1.53 WPT--Waypoint

(*regular object class*) A Waypoint is a predetermined position on the surface of the Earth used for route/instrument approach definition, or progress reporting purposes, that is defined relative to a VOR-DME station or in terms of latitude/longitude coordinates. The Waypoint's name is generally user-defined; however, when a Waypoint coincides with an ATC_Named_Waypoint, the Waypoint shall be named identically. "Waypoint" is a generally recognized term of the aviation community. Note, however, that in general usage the term "waypoint" may refer to both Waypoints and Waypoints on Flight Plans (where the meaning is clear from the context).

Summary

Primary Identifier:	wpt-1-Name
Process Model:	*present*
Attribute:	wpt-1-Latitude
Attribute:	wpt-1-Location_Description
Attribute:	wpt-1-Longitude
Attribute:	wpt-1-Magnetic_Variation
Attribute:	wpt-1-Name
Attribute:	wpt-1-Type-R17
Process:	Create_Waypoint

Definitions

wpt-1-Latitude
(*attribute*)

domn-Latitude.

wpt-1-Location_Description
(*attribute*)

A short English description of the location of the waypoint, typically the name of a nearby city.

Type:	String;

wpt-1-Longitude
(*attribute*)

domn-Longitude.

wpt-1-Magnetic_Variation
(*attribute*)

domn-Magnetic_Variation.

wpt-1-Name
(*attribute*)

A unique identifier for the Waypoint. If the Waypoint is an official ATC_Named_Waypoint (i.e., a Navigational Aid or other officially named point), the official name shall be used; If the Waypoint is defined by the pilot via reference to existing Waypoint(s) (e.g., SPO/30/20 or SPO/180/EAT/270), then the pilot-entered reference shall be used. ("SPO/30/20" and "SPO/180/EAT/270", respectively, would be the names assigned to the example Waypoints.)

Type:	String;

wpt-1-Type-R17

(subtype indication)

The class of Waypoint: is it ATC (officially) defined or just defined within this system (by pilot or company)? (This points the way to the subtype object classes, which record supplemental information appropriate to each class of Waypoint.)

Type:	Subtype Indication;
Objects:	ATCWPT, PCWPT;

Create_Waypoint

(process)

This process allows the creation and deletion of Waypoints. If the Waypoint is an official ATC_Named_Waypoint, then the official name shall be used. If the Waypoint is defined by the pilot via reference to existing Waypoint(s) (e.g., SPO/30/30 or SPO/180/EAT/270), then the pilot-entered reference shall be used as the name. ("SPO/30/30" and "SPO/180/EAT/270", respectively, would be the names assigned to the example pilot-defined Waypoints.) Likewise, if the Waypoint is defined by the pilot via latitude and longitude coordinates, then these coordinates shall be used as the name. Note that pilots may not create ATC_-Named_Waypoints.

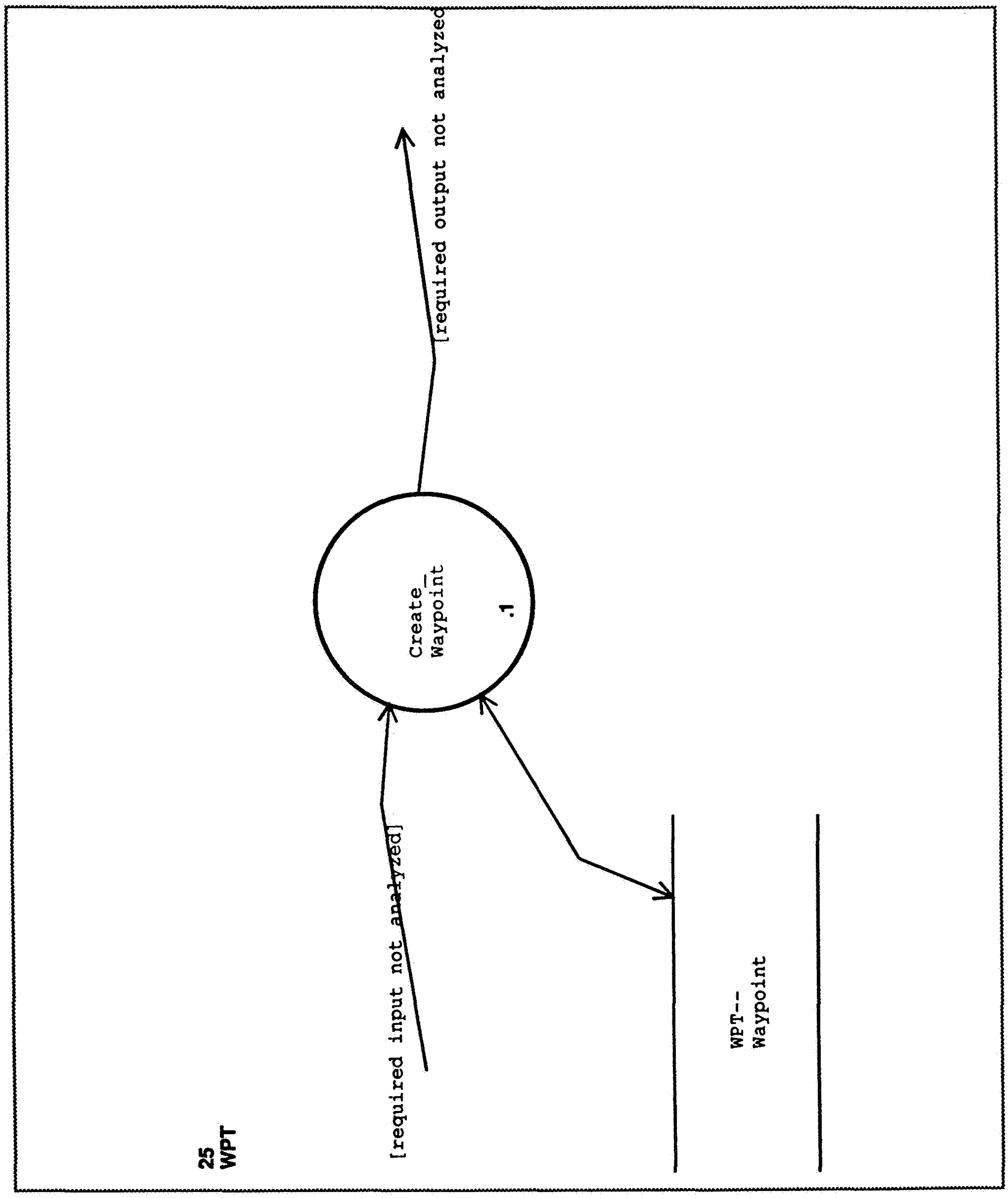


Figure 6.28 Process Model Diagram for WPT

6.1.54 WSS--Waypoint_ona_SID_or_STAR

(*regular object class*) A Waypoint_ona_SID_or_STAR arises out of the association of a particular Waypoint with a particular SS. It captures the information relevant to such an association. (The same Waypoint on a different SID or STAR may be assigned different data values.)

"Waypoint_ona_SID_or_STAR" is a somewhat artificial term created for the sake of clarity in this analysis; in general usage, the term "Waypoint" may refer to both Waypoints and Waypoints on SIDs/STARs (where the meaning is clear from the context). For that matter, the term Waypoint may refer to a Waypoint on a Flight Plan (see WFP).

Summary

Primary Identifier:	wss-1-SID-STAR_Name-R08B + wss-1-Waypoint_Name-R08A
Alias:	Waypoint_Enrollment_onan_SS, Enrollment_onan_SS;
Attribute:	wss-1-Airspeed_Constraint
Attribute:	wss-1-Airspeed_Value
Attribute:	wss-1-Altitude_Constraint
Attribute:	wss-1-Altitude_Value
Attribute:	wss-1-SID-STAR_Name-R08B
Attribute:	wss-1-Waypoint_Name-R08A

Definitions

wss-1-Airspeed_Constraint
(*attribute*)

This constraint must be continually met during the whole of waypoint acquisition. It applies to the wss-1-Airspeed_Value.

> domn-Leg_Constraint.

wss-1-Airspeed_Value
(*attribute*)

This defines the airspeed for the aircraft for the whole of waypoint acquisition. The attribute wss-1-Airspeed_Constraint defines what the aircraft must do with this value.

> domn-Airspeed.

wss-1-Altitude_Constraint
(*attribute*)

This constraint must be met just by the time the aircraft is actually at the waypoint. It applies to the wss-1-Altitude_Value.

> domn-Point_Constraint.

wss-1-Altitude_Value
(attribute)

This defines the altitude for the aircraft just when actually at the waypoint. The attribute wss-1-Altitude_Constraint defines what the aircraft must do with this value.

domn-Altitude_Above_Mean_Sea_Level.

wss-1-SID-STAR_Name-R08B
(referential attribute)

The id of the SS which the named Waypoint is on by means of this WSS.

ss-1-Name.

wss-1-Waypoint_Name-R08A
(referential attribute)

The id of the Waypoint which is on the named SS by means of this WSS.

wpt-1-Name.

6.2 Definitions of Common Domains

Common domains are created whenever two or more attributes share a common domain of values, due to an *essential* similarity (not a coincidence). This is done to follow the guideline, "Document each fact in just one place." In these cases, the group of similar attributes simply *reference* the common domain definition by its name, rather than repeat the shared *domain specification*. See Section 5.6.2 for conventions of common domains.

domn-Acceleration

(*common domain*)

Type:	Numeric;
Range:	from <> to <> feet per second per second;
Accuracy:	<>;

domn-Aileron_Position

(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Airspeed

(*common domain*)

Type:	Numeric;
Range:	from <> to <> feet per second;
Accuracy:	<>;

domn-Altitude_Above_Ground_Level

(*common domain*)

Type:	Numeric;
Range:	from 0 to <> feet;
Accuracy:	to <> feet;

domn-Altitude_Above_Mean_Sea_Level

(*common domain*)

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	to <> feet;

domn-Bank_Angle

(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Boolean
(common domain)

Type:	Enumeration;
Values:	True, False;

domn-Column_Position
(common domain)

Type:	Numeric;
Range:	from <> to <> inches;
Accuracy:	to <> inches;

domn-Elevator_Position
(common domain)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Engine_Pressure_Ratio
(common domain)

Type:	Numeric;
Range:	from <> to <>;
Accuracy:	to <>;

domn-Engine_RPM
(common domain)

Type:	Numeric;
Range:	from <> to <> [units];
Accuracy:	to <> [units];

domn-Exhaust_Gas_Temperature
(common domain)

Type:	Numeric;
Range:	from <> to <> [units];
Accuracy:	to <> [units];

domn-Engine_Throttle_Position
(common domain)

Type:	Numeric;
Range:	from 0 to 100 percent;
Accuracy:	<>;

domn-Flight_Path_Angle
(common domain)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Ground_Track_Angle
(*common domain*)

Type:	Numeric;
Range:	from 0 to 360 degrees;
Accuracy:	to <> degrees;

domn-Latitude
(*compound common domain*)

	domn-Latitude_Degrees
+	domn-Latitude_Hemisphere.

domn-Latitude_Degrees
(*common domain*)

Type:	Numeric;
Range:	from 0 to 90 degrees;
Accuracy:	<>;

domn-Latitude_Hemisphere
(*common domain*)

Type:	Enumeration;
Values:	North, South;

domn-Leg_Constraint
(*common domain*)

A "leg constraint" is a constraint which must be met during the entire acquisition of a specified point. The constraint applies to a specified value of a named variable.

Type:	Enumeration;
Values:	Must Stay At, Must Stay At or Above, Must Stay At or Below,

domn-Longitude
(*compound common domain*)

	domn-Longitude_Degrees
+	domn-Longitude_Hemisphere.

domn-Longitude_Degrees
(*common domain*)

Type:	Numeric;
Range:	from 0 to 180 degrees;
Accuracy:	tbd;

domn-Longitude_Hemisphere
(*common domain*)

Type:	Enumeration;
Values:	East, West;

domn-Magnetic_Variation

(*common domain*)

An amount of variation between actual and measured magnetic north; a correction factor to be applied to compass readings.

Type:	Numeric;
Range:	from -180 to +180 degrees;
Accuracy:	to <> degrees;

domn-Pitch

(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	<>;

domn-Point_Constraint

(*common domain*)

A "point constraint" is a constraint which must be met just by the time the aircraft is actually at a specified point. It need not be true during the acquisition of that point. The constraint applies to a specified value of a named variable.

Type:	Enumeration;
Values:	Must be At, Must be At or Above, Must be At or Below;

domn-Radio_Frequency

(*common domain*)

Type:	Numeric;
Range:	from <> to <> MHz;
Accuracy:	to <> MHz;

domn-Rudder_Pedal_Position

(*common domain*)

Type:	Numeric;
Range:	from <> to <> [units];
Accuracy:	to <> [units];

domn-Rudder_Position

(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Stabilizer_Position

(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Temperature

(*common domain*)

Type:	Numeric;
Range:	from <> to <> [units];
Accuracy:	to <> [units];

domn-Three_D_Position

(*compound common domain*)

A position over the surface of the earth.

	domn-Latitude
+	domn-Longitude
+	domn-Altitude_Above_Mean_Sea_Level.

domn-Time_of_Day

(*compound common domain*)

	domn-Time_of_Day_Hour
+	domn-Time_of_Day_Minutes.

domn-Time_of_Day_Hour

(*common domain*)

Type:	Integer;
Range:	from 0 to 23;

domn-Time_of_Day_Minutes

(*common domain*)

Type:	Integer;
Range:	from 0 to 59;

domn-Two_D_Position

(*compound common domain*)

A position on the surface of the earth.

	domn-Latitude
+	domn-Longitude.

domn-Wheel_Position

(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-X_Offset
(*common domain*)

This is the east/west component of a distance from a point over the surface of the earth. A positive value indicates displacement Eastward.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	to <> feet;

domn-X_Offset_From_Azimuth_Broadcast_Location
(*common domain*)

This is an X_Offset from the location of an Azimuth antenna.

domn-X_Offset.

domn-X_Offset_From_Runway_Threshold
(*common domain*)

This is an X_Offset from the threshold of a runway at an airport.

domn-X_Offset.

domn-Y_Offset
(*common domain*)

This is the north/south component of a distance from a point over the surface of the earth. A positive value indicates displacement Northward.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	to <> feet;

domn-Y_Offset_From_Azimuth_Broadcast_Location
(*common domain*)

This is a Y_Offset from the location of an Azimuth antenna.

domn-Y_Offset.

domn-Y_Offset_From_Runway_Threshold
(*common domain*)

This is a Y_Offset from the threshold of a runway at an airport.

domn-Y_Offset.

domn-Yaw_Angle
(*common domain*)

Type:	Numeric;
Range:	from <> to <> degrees;
Accuracy:	to <> degrees;

domn-Z_Offset
(*common domain*)

This is the vertical component of a distance from a point over the surface of the earth. A positive value indicates displacement skyward.

Type:	Numeric;
Range:	from <> to <> feet;
Accuracy:	to <> feet;

domn-Z_Offset_From_Azimuth_Broadcast_Location
(*common domain*)

This is a Z_Offset from the location of an Azimuth antenna.

domn-Z_Offset.

domn-Z_Offset_From_Runway_Threshold
(*common domain*)

This is a Z_Offset from the threshold of a runway at an airport.

domn-Z_Offset.

6.3 System Level Models

This sub-section documents the system level models (see 5.3.2): the Object Class Relationship Model and the Object Class Communication Model.

6.3.1 Object Class Relationship Model

The existence relationships among object classes are documented via the Object Class Relationship Diagram (OCRD) and, where further clarification is needed, with textual definitions. In the case of the TSRV flight control system, the OCRD suffices to specify the relationships. Figure 6.1 is a fold-out D-size plot of the OCRD.

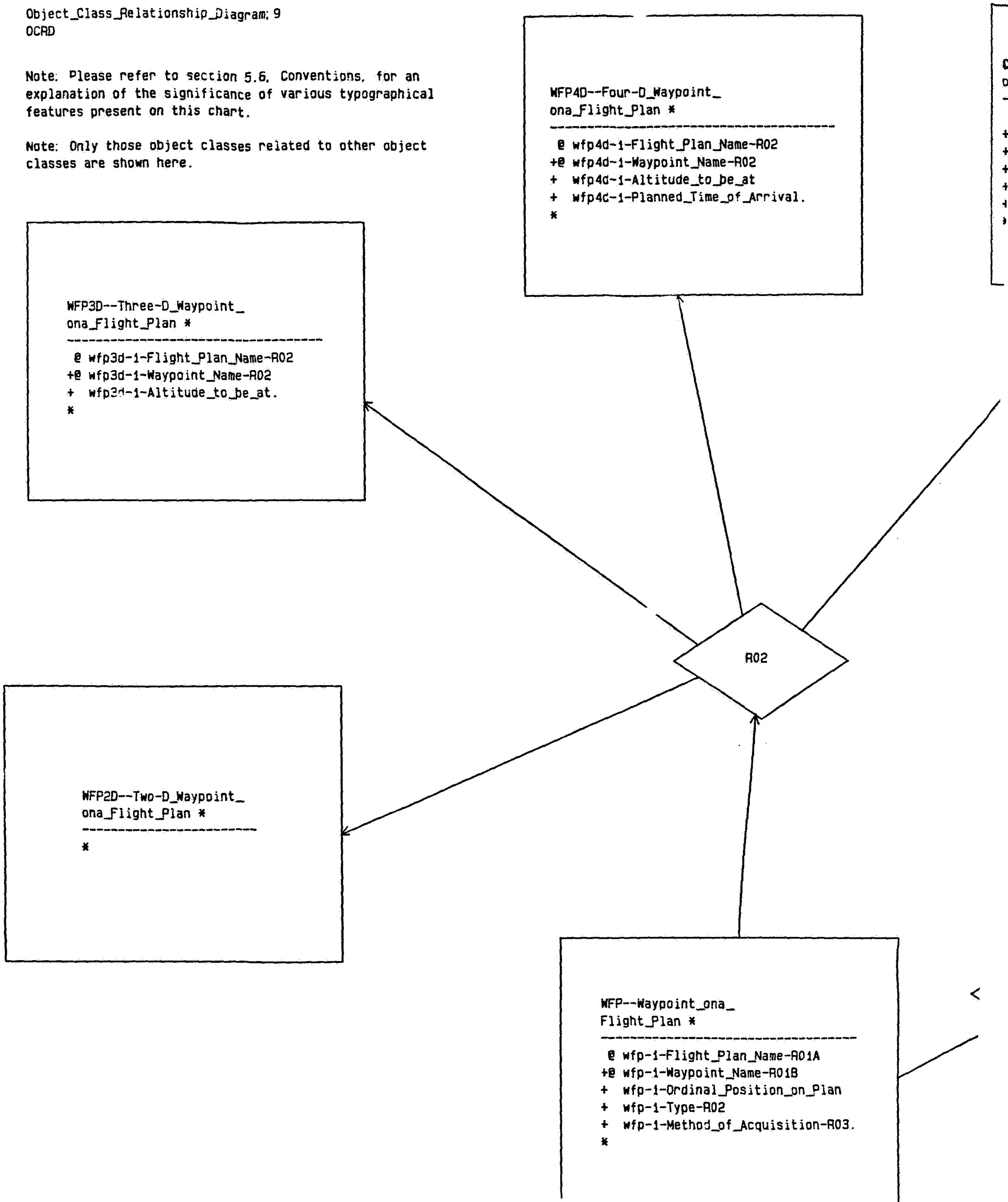
Please note that a complete understanding of the relationships depicted in the OCRD depends on one's understanding of the graphical conventions employed. Refer to 5.6.

FOLDOUT FRAME

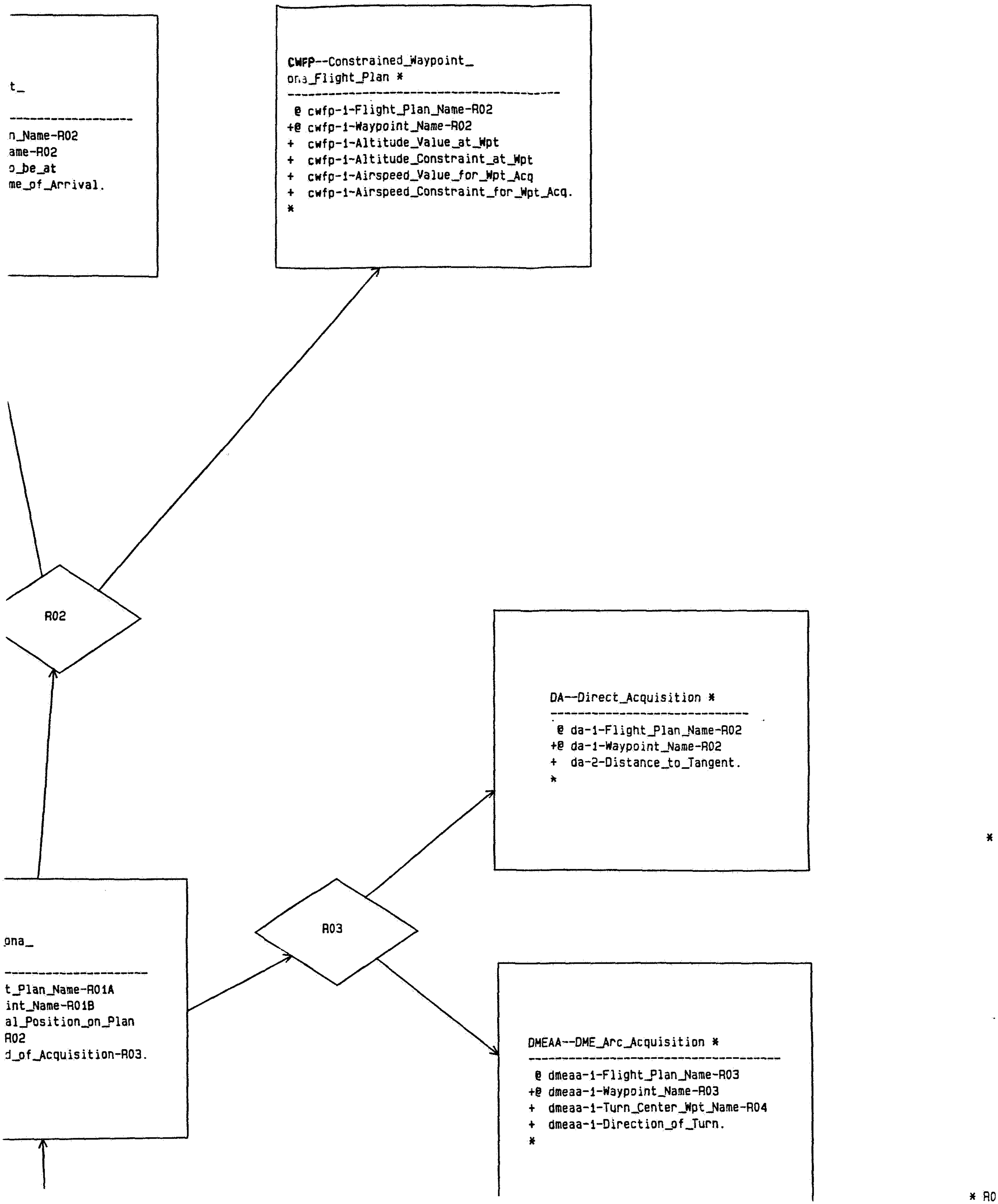
Object_Class_Relationship_Diagram: 9
OCRD

Note: Please refer to section 5.6, Conventions, for an explanation of the significance of various typographical features present on this chart.

Note: Only those object classes related to other object classes are shown here.

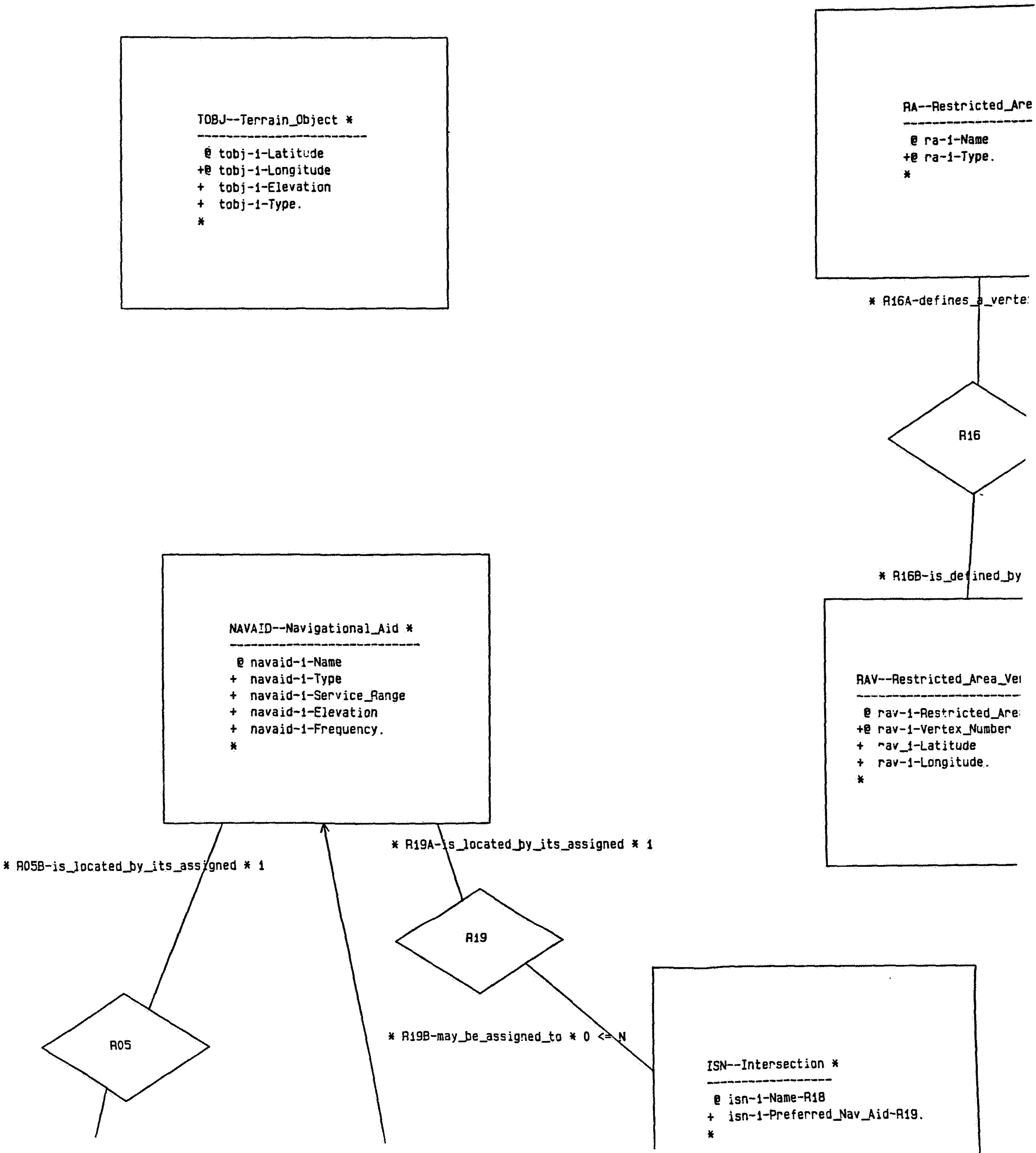


FOLDOUT FRAME 2



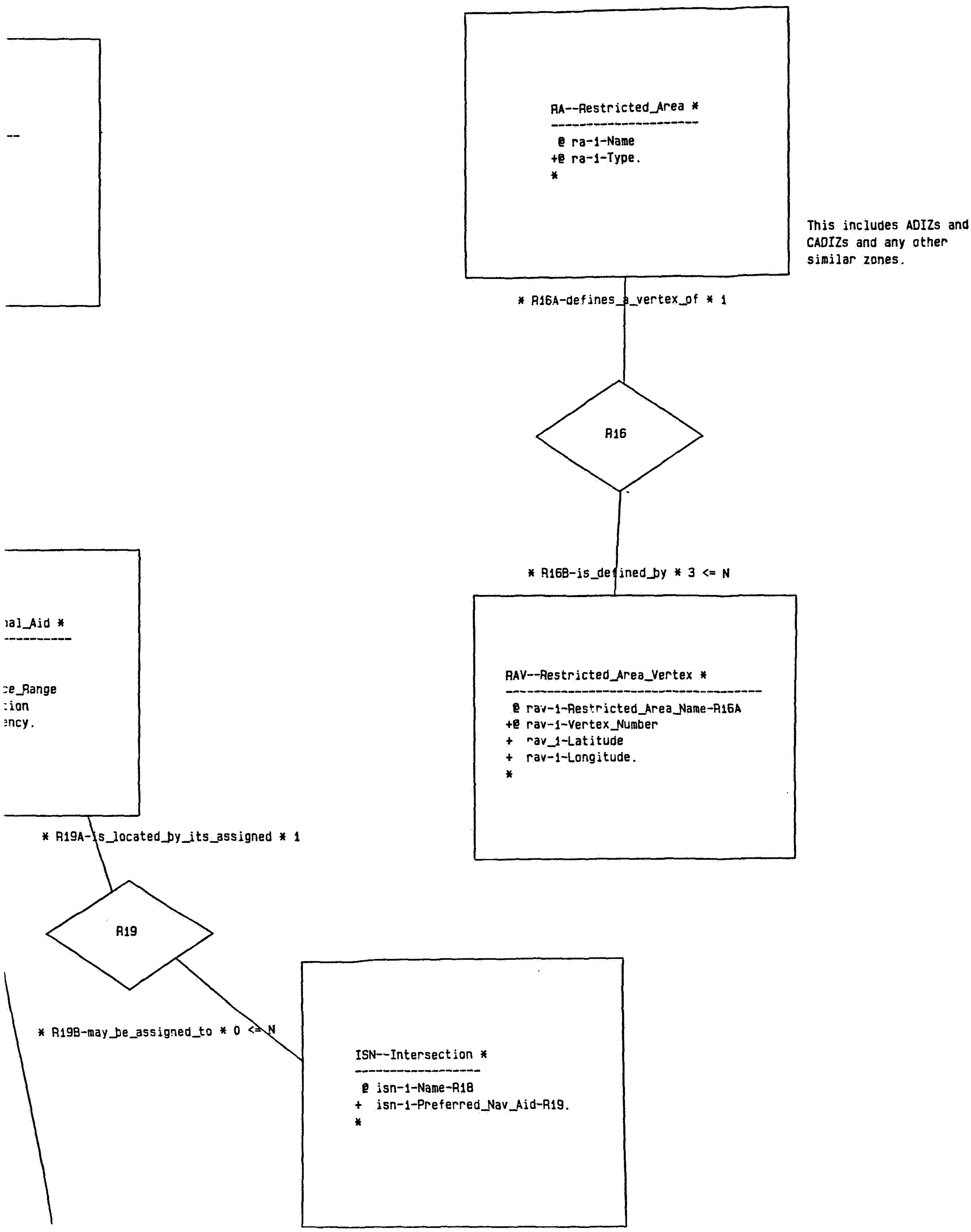
3

FOLDOUT FRAME



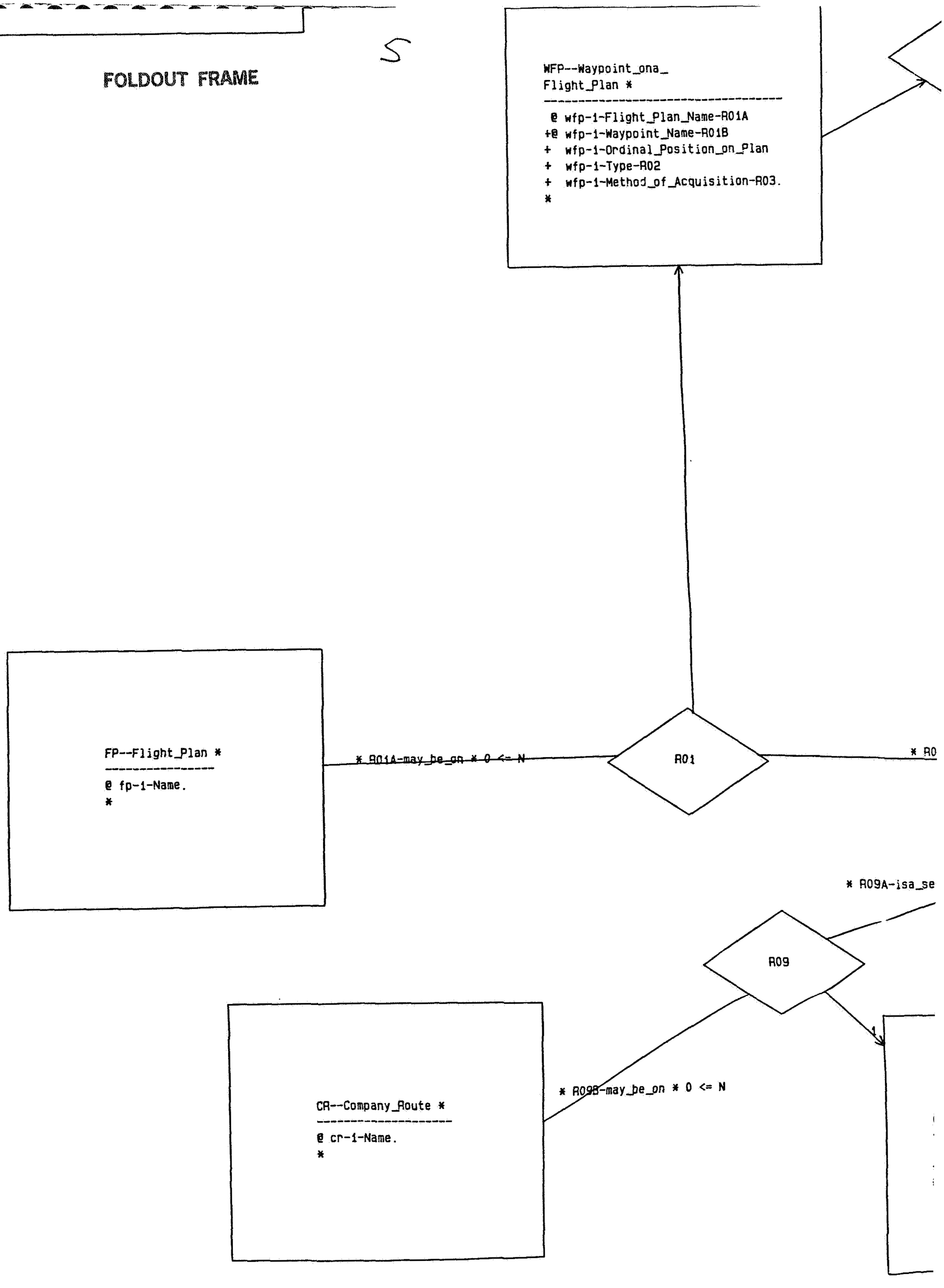
4

FOLDOUT FRAME

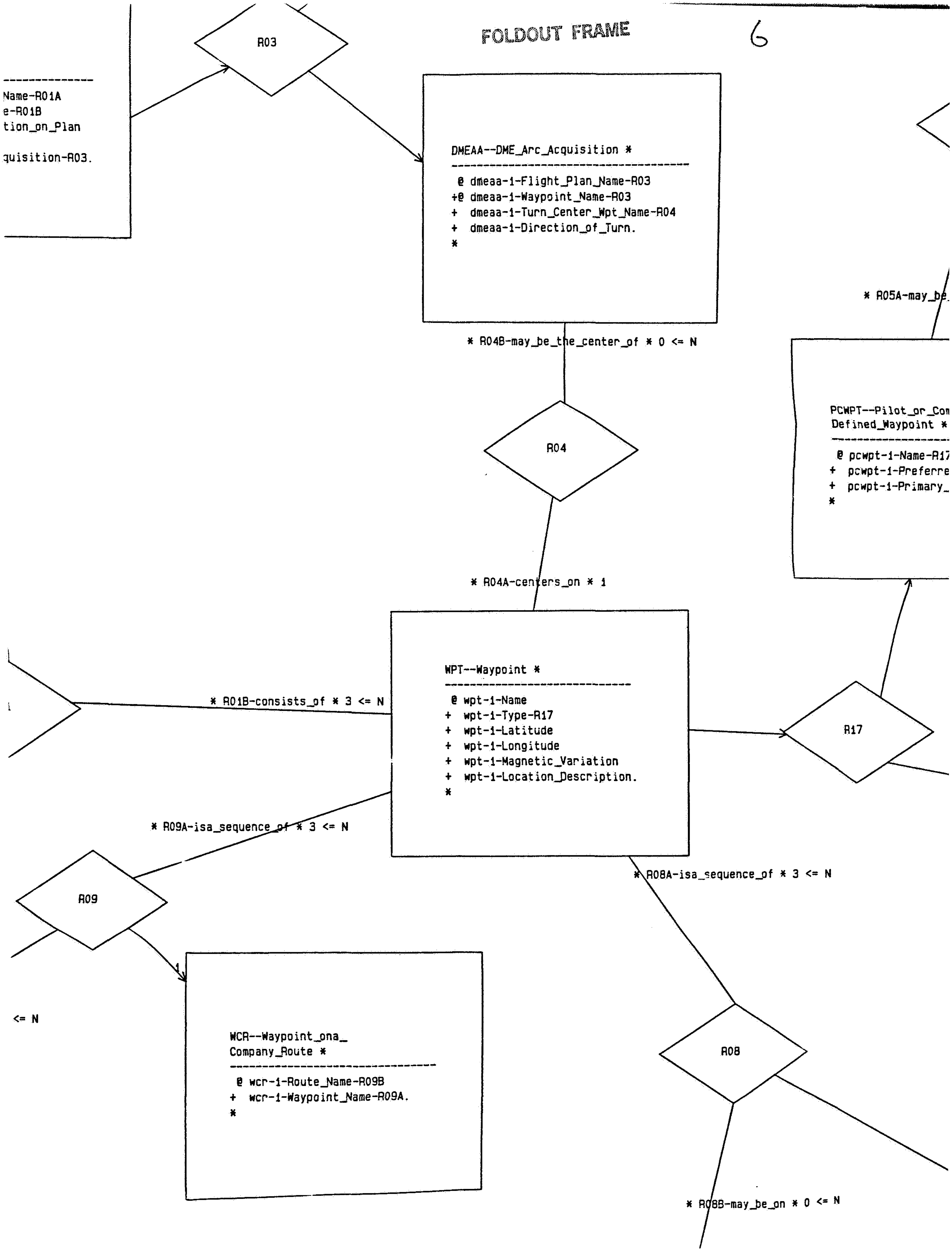


FOLDOUT FRAME

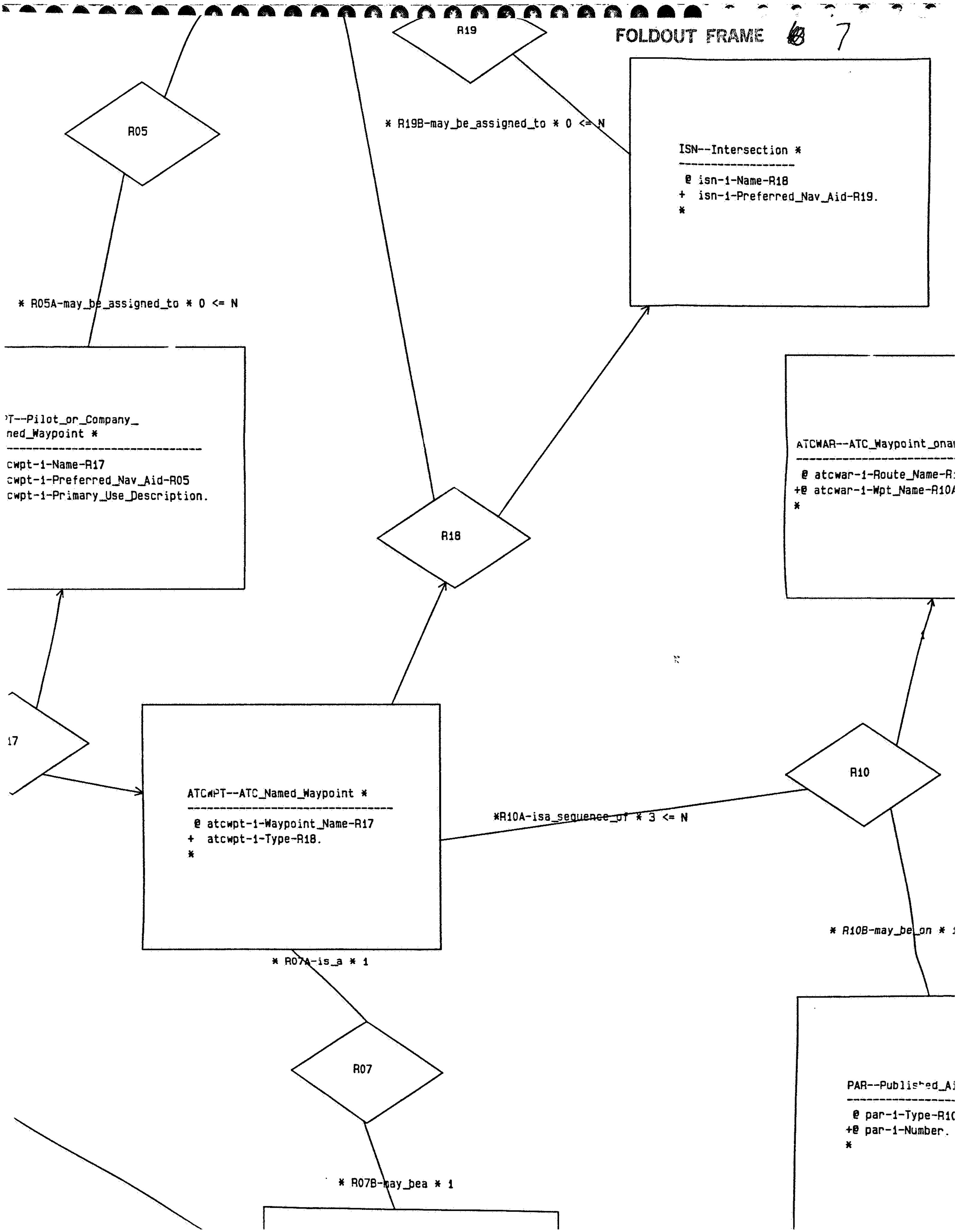
5



FOLDOUT FRAME 6

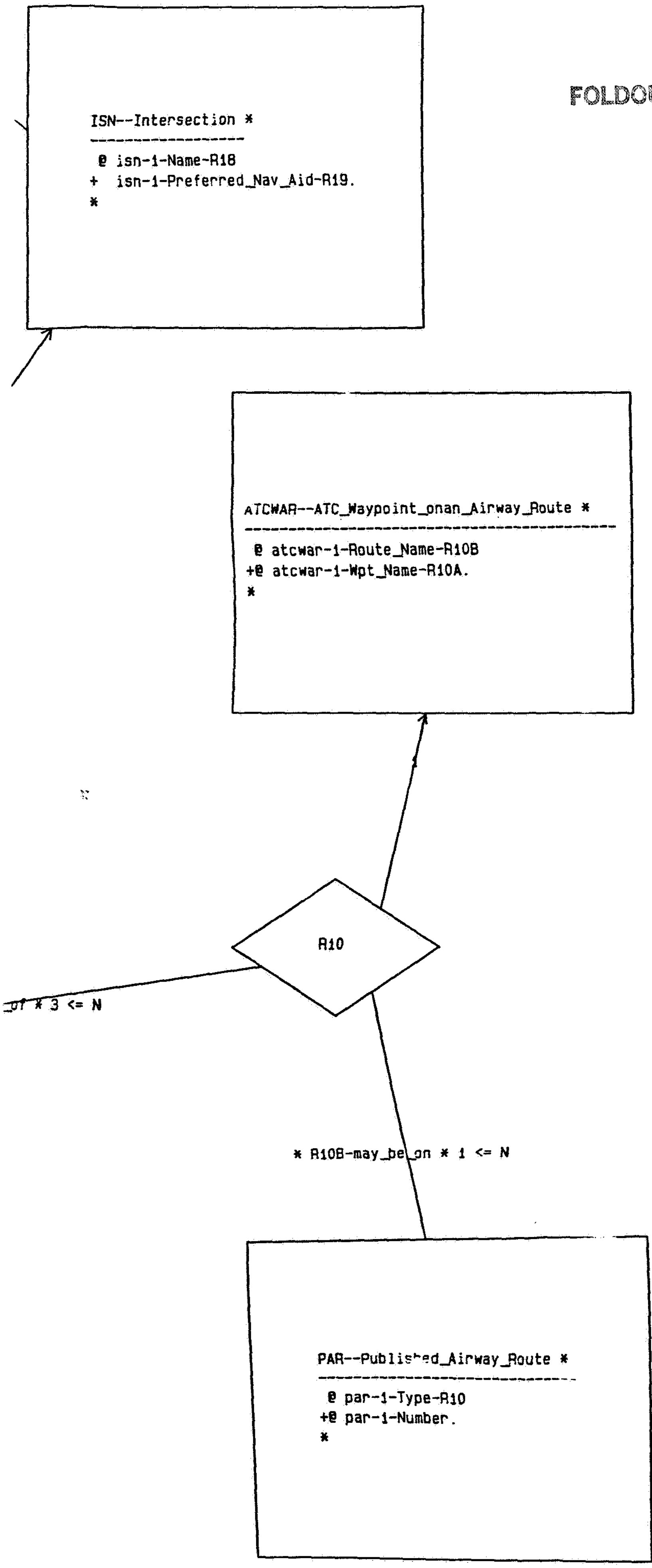


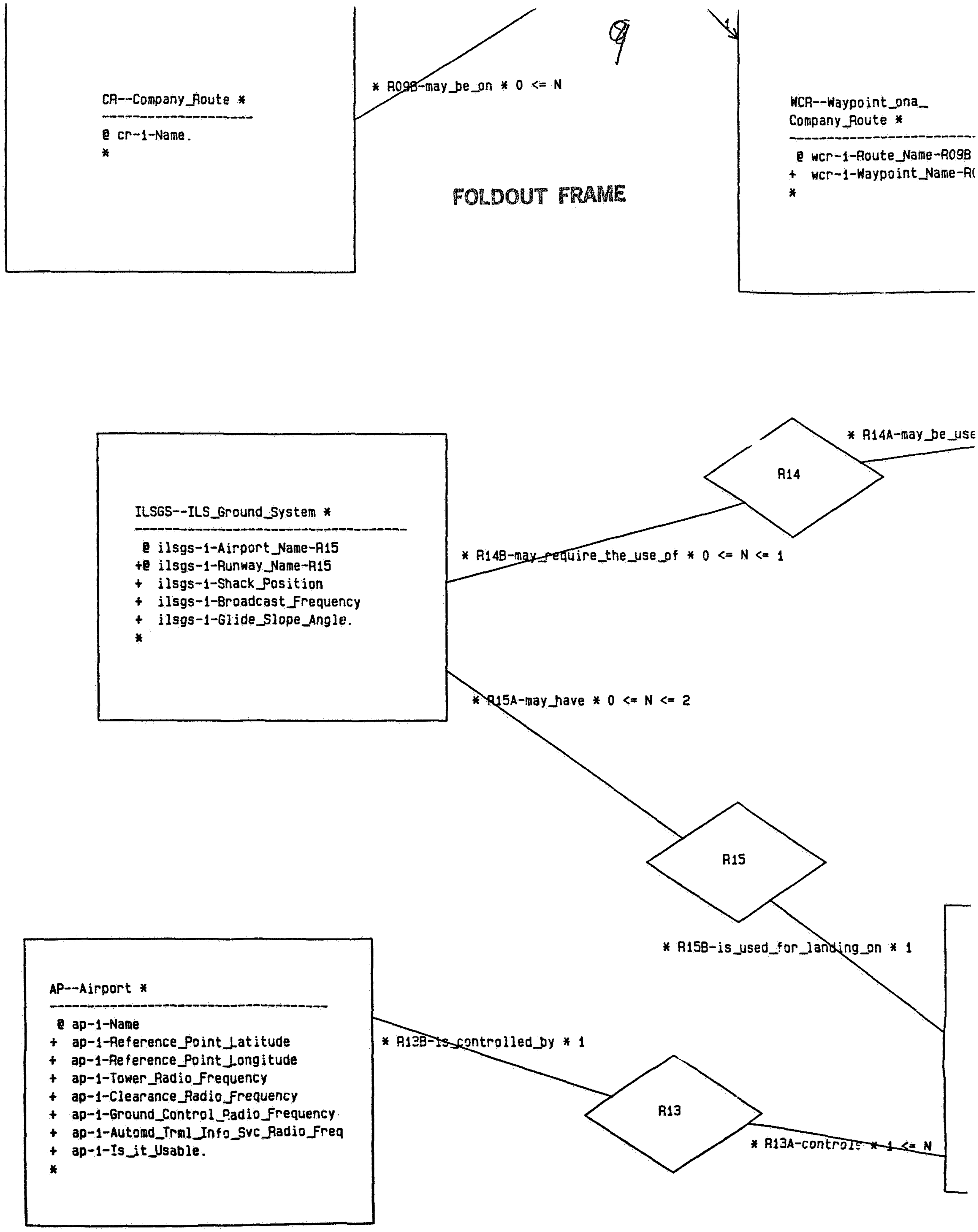
FOLDOUT FRAME 7

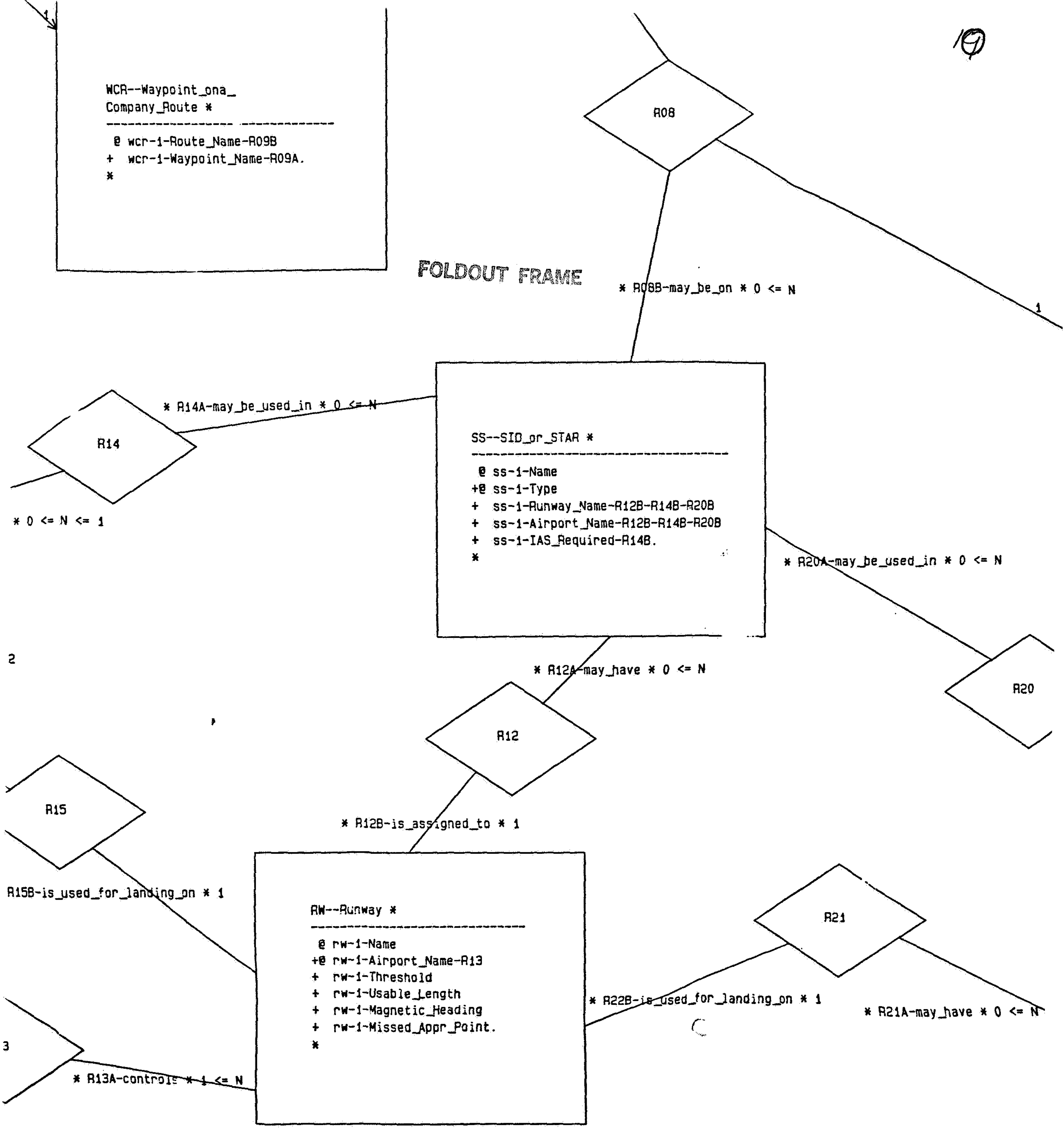


8

FOLDOUT FRAME



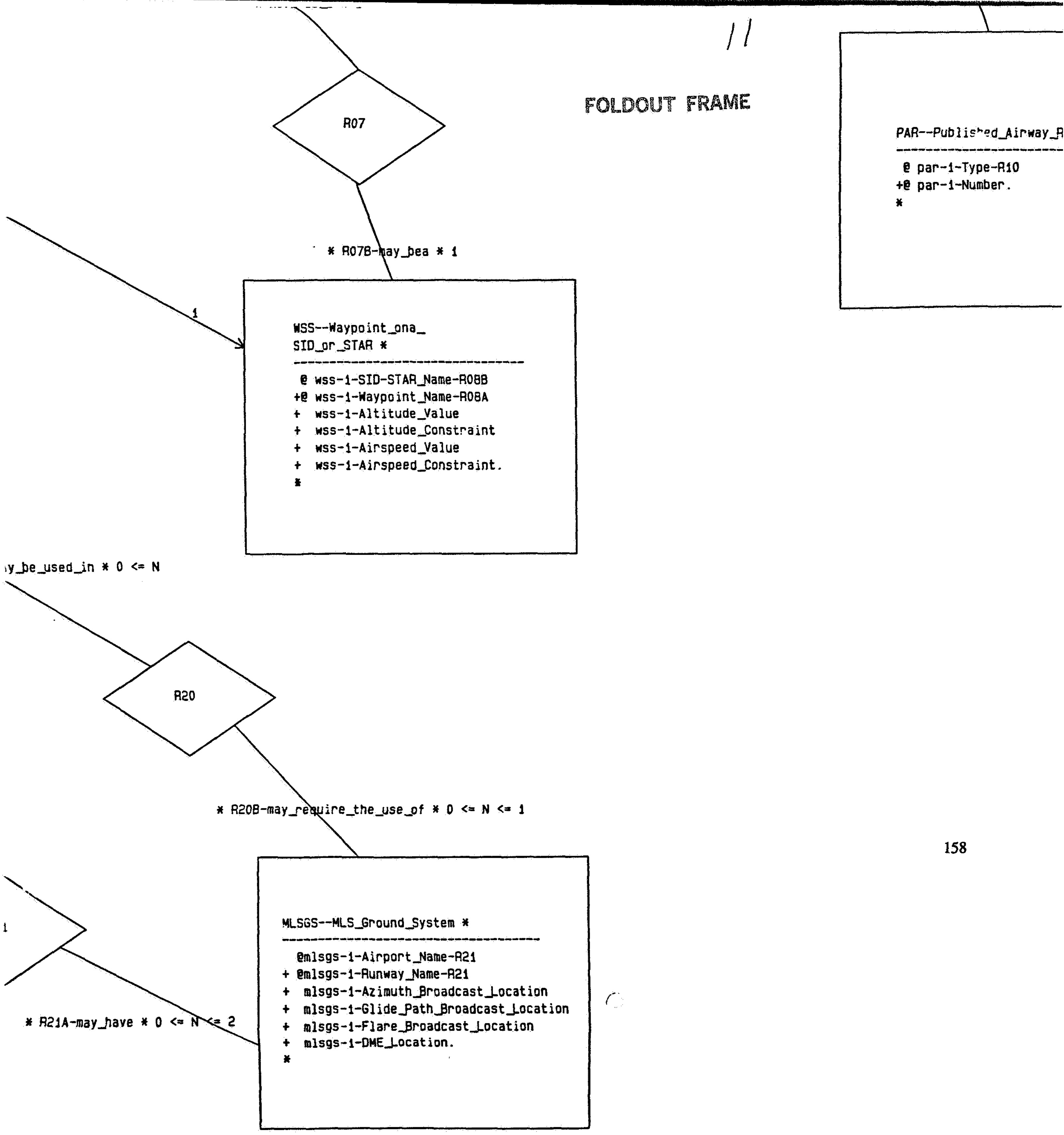

CR--Company_Route *
@ cr-1-Name.
*
* R09B-may_be_on * 0 <= N
FOLDOUT FRAME
WCR--Waypoint_ona_
Company_Route *
@ wcr-1-Route_Name-R09B
+ wcr-1-Waypoint_Name-R(
*
R14
* R14A-may_be_use
ILSGS--ILS_Ground_System *
@ ilsgs-1-Airport_Name-R15
+@ ilsgs-1-Runway_Name-R15
+ ilsgs-1-Shack_Position
+ ilsgs-1-Broadcast_Frequency
+ ilsgs-1-Glide_Slope_Angle.
*
* R14B-may_require_the_use_of * 0 <= N <= 1
* R15A-may_have * 0 <= N <= 2
R15
* R15B-is_used_for_landing_on * 1
AP--Airport *
@ ap-1-Name
+ ap-1-Reference_Point_Latitude
+ ap-1-Reference_Point_Longitude
+ ap-1-Tower_Radio_Frequency
+ ap-1-Clearance_Radio_Frequency
+ ap-1-Ground_Control_Radio_Frequency
+ ap-1-Automd_Trml_Info_Svc_Radio_Freq
+ ap-1-Is_it_Usable.
*
* R13B-is_controlled_by * 1
R13
* R13A-controls * 1 <= N



WCR--Waypoint_ona_
Company_Route *
@ wcr-1-Route_Name-R09B
+ wcr-1-Waypoint_Name-R09A.
*
R08
FOLDOUT FRAME
* R08B-may_be_on * 0 <= N
1
R14
* R14A-may_be_used_in * 0 <= N
* 0 <= N <= 1
SS--SID_or_STAR *
@ ss-1-Name
+@ ss-1-Type
+ ss-1-Runway_Name-R12B-R14B-R20B
+ ss-1-Airport_Name-R12B-R14B-R20B
+ ss-1-IAS_Required-R14B.
*
* R20A-may_be_used_in * 0 <= N
2
* R12A-may_have * 0 <= N
R12
R20
R15
* R12B-is_assigned_to * 1
R15B-is_used_for_landing_on * 1
RW--Runway *
@ rw-1-Name
+@ rw-1-Airport_Name-R13
+ rw-1-Threshold
+ rw-1-Usable_Length
+ rw-1-Magnetic_Heading
+ rw-1-Missed_Appr_Point.
*
R21
* R22B-is_used_for_landing_on * 1
* R21A-may_have * 0 <= N
3
* R13A-controls * 1 <= N

11

FOLDOUT FRAME



12

```
PAR--Published_Airway_Route *
------------------------------
 @ par-i-Type-R10
+@ par-i-Number.
*
```

FOLDOUT FRAME

Figure 6.29 Object Class Relationship Diagram

6.3.2 Object Class Communication Model

The Object Class Communication Model (OCCM) is specified by one or more Object Class Communication Diagrams (OCCDs) which summarize the data and events transferred between object classes. In the case of the TSRV flight control system, the approach was to develop multiple communication diagrams, each focusing on a related subset of the communications in the system. In the analysis strategy used (see 5.4), the development of the communication models is a simple matter of connecting the inputs and outputs of object classes and does not identify new information. In order to maximize the scope of analysis on this project, the development of OCCDs was given a low priority, and only an example OCCD was generated. The OCCD shown in Figure 6.2 specifies the communication among the objects relating to the higher level flight modes.

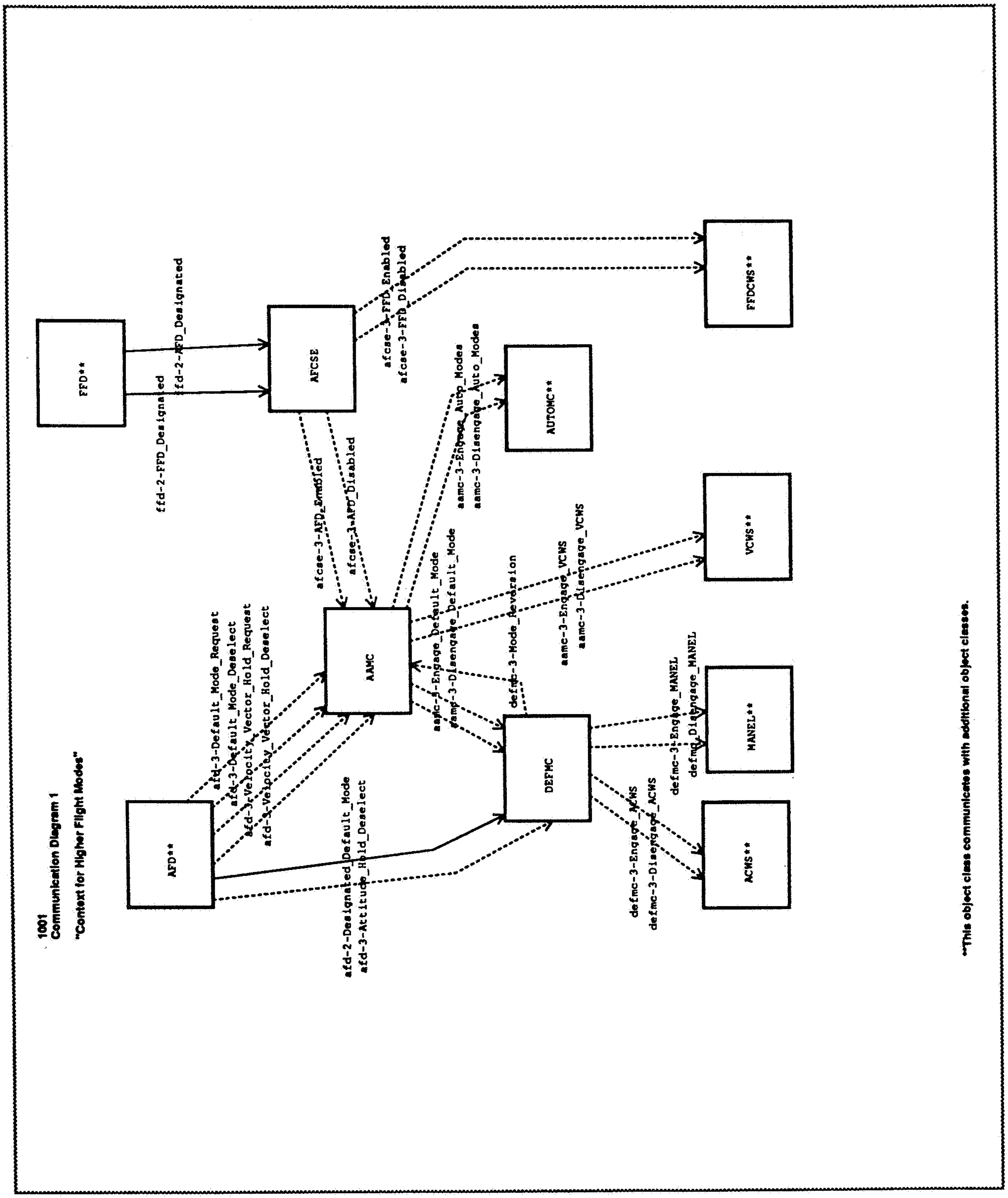


Figure 6.30 Object Class Communication Diagram

6.4 Index of Identifiers

This sub-section is intended to aid in determining where an identifier is defined or referenced. Here "identifier" refers to the names of object classes, attributes, derived attributes, events, processes and common domains. This index includes entries for every significant piece of an identifier. For instance, the attribute "althmc-1-Selected_Altitude" appears in the index under its full name as well as "1-Selected_Altitude", "Selected_Altitude" and "Altitude".

7.0 SUMMARY, RECOMMENDATIONS, AND CONCLUSIONS

A definition of a subset of the NASA Transport System Research Vehicle (TSRV) flight control system *essential* requirements was developed using the Boeing developed Object Oriented Analysis (OOA) requirements analysis methodology.

The scope of the analysis was limited to the functionality included in the Flight Management/Flight Control (FM/FC) computer which hosts the navigation, guidance, and flight control software. In terms of depth, a significant portion of the essential requirements for the FM/FC software was specified with Data, Behavior, and Process Models. Some requirements need further analysis to be fully specified. Such requirements were captured via pseudo external object classes which serve as placeholders until they can be thoroughly analyzed.

Follow-on research activities might begin with extending the scope of the analysis in terms of both breadth and depth. In regards to depth, it would be beneficial to analyze the FM/FC software further to completely ferret out the essential requirements relating to portions of the flight control laws, the autoland capability, and the sensor data requirements. This process may result in the specification of additional object classes and/or modifications of those already identified.

In regards to breadth, additional analysis might address the requirements for the on-line simulated airplane capability, the interfaces to other research flight deck systems (e.g., the IC, HOLD, and RUN operating system modes), and the displays.

The specification of the display system requirements could be addressed in at least two ways. One approach would be to perform an analysis of the Displays host computer software functions similar to that of the analysis documented in this report. (This could be extended to include the Sperry Color Display System if necessary.) The essential functionality could be integrated appropriately with the FM/FC functionality specified in this report or as extended per the above recommendations.

Alternatively, the requirements implemented in the displays host could be captured in a user interface document where the pilot(s) and the other inflight test personnel are viewed as "users" of the system. In this approach, the various pilot display formats and real-time data acquisition capabilities are considered to be a result of non-essential reformatting of the FM/FC essential data and could therefore be appropriately documented in a user interface document.

The goal of another potential follow-on activity might be to formulate a more rigorous mapping of the essential system requirements to the existing implementation. This might take the form of more detailed Process Model mini-specs or merely some degree of cross-referencing of an essential system specification to the existing software documentation. Such a mapping might provide a better understanding of the current TSRV flight control system, however, a design focused specification could hinder innovative architecture development of a new system. Also, the complexity of an object-to-code mapping may limit its usefulness.

A final recommendation for future research addresses the development of alternate more advanced software architectures for modern transport flight control systems. Architecture trade studies may be conducted on the system defined in this report or on any of its enhancements described above. In particular, artificial-intelligence based real-time architectures and related concepts have shown great potential for flight control application. Studies should consider such things as the potential implementation of existing or new functions with expert systems, innovative integration of numeric and symbolic processing, the real-world technology-based requirements associated with failure modes and reliability issues, and the appropriate use of parallel and distributed processing.

In summary, the essential system requirements specification presented here offers high visibility of the TSRV flight control system and may be readily extended to capture an increased set of existing system functionality and/or may be easily modified to accommodate new features, e.g., ones using expert system concepts. The Object Oriented Analysis methodology proved to be an effective approach to distill the essential requirements of a complex, integrated system and the information centered strategy was a useful one for backing out the requirements of an existing system.

8.0 REFERENCES

1. Frankovich, Major K., Pedersen, Ken, and Bernsteen, Stanley, "Expert System Applications to the Cockpit of the '90s," NAECON '85, Dayton, OH, May 1985.

2. Summers, Paul I., "Cockpits for 2010 and Beyond," IEEE AES Magazine, February 1986.

3. Handelman, David A., and Stengel, Robert F., "An Architecture For Real-Time Rule-Based Control," Proc. 1987 American Control Conference, Minneapolis, MN, June 1987.

4. Seward, Walter D., "Task Allocation in a Distributed Computing System," Proc. 1987 First Annual Workshop on SOAR, Houston, TX, August 1987.

5. Bruce, Kevin R., "NASA B737 Flight Test Results of Total Energy Control System," Final Report, NASA CR-178285, January 1987.

6. Goodwin, A.E., "NASA 515 Flight Control System Description," D6-34279, Boeing Commercial Airplane Company, September 1976.

7. "Advanced Transport Operating System Software Upgrade: Flight Management/Flight Controls Software Description," Computer Sciences Corporation, NASA CR-181936, January 1988.

8. McMenamin, Stephen M., and Palmer, John F., Essential Systems Analysis, Yourdon Press, 1984.

9. Tockey, Steve, Hoza, Brad, and Cohen, Sam, "Object Oriented Analysis: Building on the Structured Techniques," Proc. Software Improvement Conference (sponsored by The Educational Foundation of the Data Processing Management Association), Washington D.C., November 29-30, 1990.

10. DeMarco, Tom, Structured Analysis and System Specification, Yourdon Press, 1979.

11. Ward, Paul T., and Mellor, Stephen J., Structured Development for Real-Time Systems," Volumes 1 through 3, Yourdon Press, 1985.

12. Shlaer, Sally, and Mellor, Stephen J., Object Oriented Analysis: Modeling the World in Data, Prentice Hall, 1988.

13. "**Team***work* Environment User's Guide, Release 3.0.3," ENUSE 3.03, Cadre Technologies Inc., 1988.

14. Tockey, Steve, "Using Off-the-Shelf CASE Tools to Build Object Oriented Analysis Specifications," Proc. Structured Development Forum, San Diego, CA, May 1-3, 1990.

C-3

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 1992

3. REPORT TYPE AND DATES COVERED
Contractor Report

4. TITLE AND SUBTITLE
NASA TSRV Essential Flight Control System Requirements Via Object Oriented Analysis

5. FUNDING NUMBERS
C NAS1-18027
WU 505-66-41-04

6. AUTHOR(S)
Keith S. Duffy and Bradley J. Hoza

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Boeing Commercial Airplane Group
P.O. Box 3707, M/S 6U-ER
Seattle, WA 98124-2207

8. PERFORMING ORGANIZATION REPORT NUMBER
NASA CR-189573

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Richard M. Hueschen
Langley Contacting Officer's Technical Representative: Cary R. Spitzer
Final Report - Task 15

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 08

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objective of this work is to analyze the baseline flight control system of the Transport Systems Research Vehicle (TSRV) and develop a system specification that offers high visibility of the essential system requirements in order to facilitate the future development of alternate, more advanced software architectures. The flight control system is defined to be the baseline software for the TSRV research flight deck, including all navigation, guidance, and control functions, and primary pilot displays.

The Object Oriented Analysis (OOA) methodology developed by Boeing is used to develop a system requirements definition. The scope of the requirements definition contained herein is limited to a portion of the Flight Management/Flight Control computer functionality. The development of a partial system requirements definition is documented, and includes a discussion of the tasks required to increase the scope of the requirements definition and recommendations for follow-on research.

14. SUBJECT TERMS
ATOPS TSRV; Object Oriented Analysis; essential requirements; advanced architecture

15. NUMBER OF PAGES
182

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

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.