DOMAIN ENGINEERING VALIDATION CASE STUDY:
SYNTHESIS FOR THE AIR TRAFFIC DISPLAY/COLLISION WARNING MONITOR DOMAIN

SPC-92050-CMC

VERSION 01.00.03

NOVEMBER 1992

DOMAIN ENGINEERING VALIDATION CASE STUDY: SYNTHESIS FOR THE AIR TRAFFIC DISPLAY/COLLISION WARNING MONITOR DOMAIN

SPC-92050-CMC

VERSION 01.00.03

NOVEMBER 1992

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM SERVICES CORPORATION
under contract to the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

CONTENTS

FIGURES

TABLES

.

This page intentionally left blank.

ACKNOWLEDGEMENTS

This page intentionally left blank.

1. INTRODUCTION

In 1991, the Consortium began a case study applying the Synthesis reuse-driven software development process to the Air Traffic Display/Collision Warning Monitor (ATD/CWM) domain. This case study had the following goals:

- Gain experience in the application of a Synthesis reuse process.

- Refine and validate the activity descriptions of the Synthesis guidebook.

- Illustrate the practice of a Synthesis process using specific methods.

- Provide examples of Synthesis work products.

A domain of ATD/CWM systems was chosen with concern for the following criteria:

- Be relevant to member company problems.

- Be a realistic problem.

- Have both behavioral and informational variations.

- Have real-time constraints.

- Represent an embedded system.

- Require domain knowledge that is readily available.

The ATD/CWM domain satisfies these criteria as follows (Horne 1989; Nordwall 1991, Connes 1992):

- Commercial systems in this domain are being built today. B.F. Goodrich FlightSystems (formerly known as Foster AirData) is building a collision warning system called CWS691. This system was borne out of a U.S. Navy contract in which Foster AirData developed a NA-VAL Aircraft Collision Warning System (NACWS). Bendix/King Air Transport Avionics Division of Allied-Signal Aerospace Company, Honeywell, and Collins are building a Traffic Alert and Collision Avoidance System (TCAS).

- Variations exist within these systems. For example, there are three types of TCAS: I, II, and III. TCAS I depicts locations of transponder-equipped aircraft that may pose a collision threat. TCAS II goes further showing the intruding aircraft's altitude and whether it is flying level, climbing, or descending. It also issues voice and visual commands telling the pilot to climb, descend, or fly level to avoid intruders. TCAS III does everything TCAS II does plus it issues commands to turn left or right.

- These are embedded systems. They reside completely within the aircraft and are subject to weight and size constraints.

- These are real-time systems. They must detect potential collisions and respond to these potential collisions in real time.

- Systems are customized for different customers. For example, United Airlines had the scan heights for its systems customized for the climb, descent, and cruise flight phases.

This domain gives the Consortium the basis for exploring many of the issues that could reduce the effectiveness or acceptability of Synthesis if their implications are not sufficiently considered. However, the case study, being limited in effort and based on limited in-house domain knowledge, does not necessarily represent a commercially viable formalization of the domain. In particular, the case study currently treats real-time issues and environmental constraints superficially. As such, this case study should be viewed as "representative only" of how the Synthesis approach organizes and applies available expertise.

The definition of the ATD/CWM domain started with an existing description of customer requirements for an ATD/CWM system. (See the Statement of Requirements in *On-Board Embedded Air Traffic Display/Collision Warning Monitor System ATD/CWM* [P.P. Texel & Co. 1987]). This specification was also the subject of an Ada-based Design Approach for Real-Time Systems (ADARTS) case study (Software Productivity Consortium 1991b). The Consortium modified the specification, somewhat, to be more reflective of actual systems, which resulted in the description shown in Appendix D. By considering possible variations in this description, a concept of a family of ATD/CWM systems arises of which the original system is one instance.

1.1 DOCUMENT PURPOSE, SCOPE, AND AUDIENCE

This case study exemplifies Synthesis guidance, as provided in the 1991 Synthesis Guidebook (SPC 1991c), and its application to the ATD/CWM domain. The Synthesis guidance will be one volume of the 1992 guidebook for reuse-driven software development processes. This document covers both domain engineering and application engineering work products. Even though the Synthesis reuse process is an iterative process, this case study will present only the work products of the final (to date) iteration. (The previous iterations are documented in Volume 2 of the 1991 Synthesis Guidebook [SPC 1991d]). This case study will provide some discussion of how these work products were refined or evolved from previous work product versions.

The case study will help line engineers and technologists understand the application of the Synthesis reuse-driven software development process by providing examples of the work products created by applying Synthesis to a particular domain constituting a business area focus. In this context, a domain is a set of applications.

1.2 DOCUMENT STRUCTURE

Sections 2 through 8 contain the ATD/CWM domain engineering work products.

Section 2, the ATD/CWM Domain Definition, establishes the scope of the ATD/CWM domain and a justification of its economic viability. The Domain Definition provides a basis for informally determining whether a system is properly within that scope. A Domain Definition consists of a Domain Synopsis, Domain Glossary, Domain Assumptions, and Domain Status.

Section 3, the ATD/CWM Decision Model, specifies the decisions that the Application Modeling Notation must allow an application engineer to make in describing an instance of the ATD/CWM domain. The Decision Model consists of decision specifications, decision groups, and decision constraints.

Section 4, the ATD/CWM Product Requirements, specifies the software requirements of members of the ATD/CWM product family. The Product Requirements also ascribe the meaning to Application Models created in accordance with the corresponding Decision Model. The Product Requirements are a parameterized description of software for a product in the domain: including implicit requirements and derived requirements.

Section 5, the ATD/CWM Process Requirements, defines a standard process that the Application Engineers follow to develop and evolve systems in the ATD/CWM domain. The Process Requirements work product consists of the Process Specification and Application Modeling Notation.

Section 6, the ATD/CWM Product Design, specifies the design of members of the ATD/CWM product family. The Product Design consists of a Product Architecture, Component Design, and Generation Design.

Section 7, the ATD/CWM Product Implementation, is an adaptable implementation of the ATD/CWM product family. The Product Implementation consists of Adaptable Components and a Generation Procedure.

Section 8, the ATD/CWM Process Support, describes the procedures and standards by which application engineers develop ATD/CWM applications (application engineering process), the automated environment which supports the effective and correct performance of the application engineering process, and documentation supporting the use of the procedures, standards, and environment.

Sections 9 and 10 describe examples of ATD/CWM application engineering products. Section 9, the ATD/CWM Application Model, captures the requirements for the ATD/CWM system described in Appendix D. An Application Model describes a deliverable system in terms of requirements and engineering decisions.

Section 10, the ATD/CWM Application Software, gives fragments of the ATD/CWM Application Software derived from the ATD/CWM Application Model described in Section 9. Application Software (both code and documentation) is derived mechanically from the Application Model to provide a capability specified by customer requirements.

Four appendixes follow the ATD/CWM work products. Appendix A describes the requirements method used to specify the Product Requirements for the ATD/CWM domain. Appendix B describes presentation paradigms required by that method for describing the form of output data. Appendix C shows, for the ATD/CWM domain, how a commercially available tool can be used in support of Domain and Application Engineering. Appendix D presents customer requirements for one ATD/CWM system.

1.3 RELATED PUBLICATIONS

The reader must be familiar with the activities, work products, and terminology of the Synthesis reuse-driven software development process as described in *Synthesis Guidebook Volume 1*

Methodology Definition (Software Productivity Consortium 1991c) before attempting to read this document. In addition to this document, the reader may want to refer to the following publications that describe certain methods used in the case study.

- *TRF2 Metaprogramming Tool User Guide* (Software Productivity Consortium 1991a) describes a metaprogramming notation and tool for creation and use of abstract components.

- *ADARTS Guidebook* (Software Productivity Consortium 1991b) describes the software design method used for systematically structuring a real-time system into concurrent tasks and packages to achieve a modifiable system.

1.4 TYPOGRAPHIC CONVENTIONS

This case study uses the following typographic conventions:

Serif font .... General presentation of information.

Capitalized Serif font .... Names of Synthesis work products.

Italicized serif font .... Publication titles.

Boldfaced serif font .... Section headings and emphasis.

`Typewriter` font .... Syntax of code.

| .... Separator for a list of alternatives.

Additional information to aid in understanding and using the case study work products is presented as *NOTES*.

Domain Engineering work products require use of a metaprogramming notation to represent variability in a product. Variability in a product means that a product will have different content depending on certain critical decisions. A metaprogramming notation allows you to describe how a product's content is determined by those decisions. A simple example of this is the use of a macroprocessor to defer a decision about the size of a data structure. Instead of making the decision on the size of the structure when the code is written (by embedding a constant), you can defer the decision by parameterizing the code and supplying the required value at compile time. A metaprogramming notation is an extension of this idea.

Boldfaced, bracketed text is used for metaprogramming notation in this document.

< boldfaced_identifier > .... A deferred decision (e.g., **< size >**). Such identifiers can be separated by dots to indicate elements of composite decisions (e.g., **< stack.type >** and **< stack.size >**). This identifier is replaced with the actual value of the decision whenever an instance of the containing work product is created.

< if predicate then > body1 [< else > body2] < endif > ....
A conditional inclusion. A predicate is an informal

truth-valued expression (i.e., one that only takes on the values true or false) defined in terms of decisions. If the predicate evaluates to true, then body1 is included in the work product. If an **else** clause is included and the predicate evaluates to false, then body2 is included in the work product.

< forall X in list > body < endfor > An iterative (repeating) inclusion. The list is an identifier for a decision that is multi-valued. This construct includes one copy of body in the work product for each value of the decision. For each copy of body included, the corresponding decision value replaces all occurrences of identifier X in that copy.

A predicate can take on different forms depending upon the nature of the truth-valued expression. The meaning of the forms commonly used in this document are explained below.

X . . . . The value of identifier **X** is the value of the predicate. Furthermore, the value of **X** can only be true or false.

X = Y . . . . The predicate value is determined by comparing the value of X with Y. If they are equal, then the predicate value is true. Otherwise, it is false.

X = {Y} . . . . X is an identifier for a composite decision. One or more elements $(x_1, x_2, ... x_n)$ of identifier X can have a value. The absence of a value for any x_i is considered a "don't care." Y is a list containing one or more elements x_i. The value of the predicate is true when the set of elements of X having a value equals the list of elements contained in Y. Otherwise, the value of the predicate is false.

X OR Y . . . . The predicate value is false if both X and Y are false. Otherwise, the value of the predicate is true.

there exists X in Y . . . . Y is an identifier for a decision that is multi-valued. The value of the predicate is true when the value of X equals at least one of the values for Y. Otherwise, the value of the predicate is false.

there exists X ∈ Y . . . . Y is an identifier for a decision that is multi-valued. The value of the predicate is true when Y has at least one value. Otherwise, the value of the predicate is false.

there exists X ∈ Y such that Z . . . . . Y is an identifier for a composite decision that is multi-valued. The truth-valued expression Z uses one or more elements of identifier X. The value of the predicate is true when there is at least one composite decision X in Y such that Z is true. Otherwise, the predicate value is false.

A body is any text that may be a part of some work product. A body may also contain nested metaprogramming constructs: if so, those constructs have to be evaluated to determine the content of the body.

TRF2 metaprogramming notation, which has a similar use but a different form, is used in some case study work products. See the *TRF2 Metaprogramming Tool User Guide* (Software Productivity Consortium 1991a) for more information on the form and use of the TRF2 metaprogramming notation.

2. ATD/CWM DOMAIN DEFINITION

1. DOMAIN SYNOPSIS

The ATD/CWM domain is a family of embedded computer systems, installed in an aircraft, to monitor air traffic within a surrounding surveillance area and to detect potential collision situations. Systems in the ATD/CWM domain monitor flight characteristics (e.g., altitude, ground_track, range) of potential threats and display the flight characteristics within the host aircraft's cockpit. Flight characteristics are obtained from messages transmitted by either potential threats or air traffic control centers. These computer systems may also identify collision situations and take actions such as displaying collision warning characteristics (e.g., location and airspeed of potential threat) and corrective action advisory messages within the host aircraft's cockpit, transmitting inter_air messages to potential threats, and transmitting advisory messages to an air traffic control center.

2. DOMAIN GLOSSARY

Many of the definitions in this glossary originate from the *AOPA's Aviation USA* (Aircraft Owners and Pilots Association 1990). The AOPA manual is derived from terms and definitions compiled by the Federal Aviation Administration (FAA) in both the Pilot/Controller Glossary section of the Airman's Information Manual and in Federal Aviation Regulation Part 1 (ASA Publications 1989). In this glossary, all terms marked with a superscript 1 appear in the AOPA; those marked with a superscript 2 are taken from *Webster's II New Riverside University Dictionary* (Webster 1984).

Advisory message	A message transmitted from a host aircraft to an air traffic control center when the ATD/CWM system detects a collision warning situation.
Aircraft[1]	Device(s) that are used or intended to be used for flight in the air and, when used in air traffic control terminology, may include the flight crew.
Aircraft_identification	Any unique aircraft identifier (e.g., transponder code or tail number).
Airspeed[1]	The speed of an aircraft relative to its surrounding air mass. The unqualified term "airspeed" means one of the following: (1) Indicated Airspeed—The speed shown on the aircraft airspeed indicator. This is the speed used in pilot/controller communications under the general term "airspeed." (2) True Airspeed—The airspeed of an aircraft relative to undisturbed air. This is the speed used primarily in flight planning and the enroute portion of a flight. When used in pilot/controller communications, it is referred to as "true airspeed" and not shortened to "airspeed."

Air traffic control[1]

A service operated by appropriate authority to promote the safe, orderly, and expeditious flow of air traffic.

Air traffic control device

To be defined.

Altitude[1]

The vertical distance height of a level, point, or object considered as a point, measured from mean sea level.

Bearing[1]

The horizontal direction to or from any point, usually measured clockwise from true north, magnetic north, or some other reference point through 360 degrees.

Climb rate

An aircraft's change in altitude per time unit.

Cockpit[2]

The space in the fuselage of an aircraft with seats for the pilot and crew.

Collision warning characteristic

A characteristic of the relationship between a host aircraft and a potential threat (e.g., time_to_intersect, range, ground_track).

Collision warning situation

A situation that arises when a potential threat has certain values of its collision warning characteristics relative to the host aircraft. A given potential threat can progress through different collision warning situations relative to the host aircraft.

Corrective action advisory message

A message displayed on the host aircraft's display describing what actions the host aircraft should take to avoid a collision warning situation.

Course[1]

The intended direction of flight in the horizontal plane measured in degrees from north.

Display device

To be defined.

Flight characteristics

Characteristics of an aircraft's flight (e.g., airspeed, ground_track, range, aircraft_identification).

Flight path[1]

A line, course, or track along which an aircraft is flying or is intended to be flown.

Ground speed[1]

The speed of an aircraft relative to the surface of the earth.

Ground_track

Ground_track is measured from the line of the aircraft to magnetic north to the horizontal component of the aircraft's track. Ground_track is measured in degrees with a resolution of one degree.

Heading[1]

Informs the pilot of the heading he should fly. The pilot may have to turn to, or continue on, a specific compass direction to comply with the instructions. The pilot is expected to turn in the shorter direction to the heading unless otherwise instructed by air traffic control.

Host aircraft

The aircraft that is monitoring other aircraft in its surveillance area.

Icon

A graphical presentation of an aircraft. Characteristics of an icon include 2-dimensional shape, color, and shading (i.e., filled in).

Inter_air message

A message transmitted from the host aircraft to a potential threat when a collision warning situation has been detected by the host aircraft.

Interrogator[1]

The ground-based surveillance radar beacon transmitter/receiver which normally scans in synchronism with a primary radar, transmitting discrete radio signals which repetitiously request all transponders, on the mode being used, to reply.

Intersection

A 3-dimensional region where two flight paths cross within a separation minima.

Mode[1]

The letter or number assigned to a specific pulse spacing of radio signals transmitted or received by ground interrogator or airborne transponder components of the air traffic control radar beacon system. Mode A (number), Mode C (number and altitude reporting), and Mode S (number, altitude reporting and a data link) are used in air traffic control.

Navigation device

To be defined.

Normal situation

A situation that is not a collision warning situation.

Potential threat

An aircraft within the host aircraft's surveillance area.

Radar device

To be defined.

Range

The distance measured from the host aircraft to another point (e.g., potential threat, air traffic control center).

Relative_bearing

Bearing of the potential threat relative to the host aircraft. Relative_bearing is measured from the ground_track of the host aircraft to the line from the host aircraft to the potential threat in the clockwise direction looking down.

Separation minima[1]

The minimum longitudinal, lateral, or vertical distances by which aircraft are spaced through the application of air traffic control procedures.

Surveillance area

The 3-dimensional area around a host aircraft that must be monitored.

Time group[1]

Four digits representing the hours and minutes from the 24-hour clock. Time groups without time zone indicators are understood to be UTC (Coordinated Universal Time); e.g., 0205. The term Zulu is used when air traffic control procedures require a reference to UTC. A time zone designator is used to indicate local time; e.g., 0205M. The end and beginning of the day are shown by 2400 and 0000, respectively.

Time_to_intersect

Describes the elapsed time in which two aircraft flight paths could possibly intersect.

Track[1]

The actual flight path of an aircraft over the surface of the earth.

Transponder[1]

The airborne radar beacon transmitter/receiver portion of the air traffic control radar beacon system which automatically receives radio signals from interrogators on the ground and selectively replies with a specific reply pulse or pulse grouped only to those interrogations being received on the mode to which it is set to respond.

Specialization Taxonomy

Aircraft
    Host aircraft
    Potential threat

Air traffic control center

Cockpit

Collision warning characteristic
    Intersection
    Range
    Relative_bearing
    Time_to_intersect

Situation
    Collision warning situation
    Normal situation

Message
    Advisory message
    Corrective action advisory message
    Inter_air message

Flight characteristics
    Aircraft_identification
    Airspeed
        Ground speed
    Altitude
    Climb rate
    Course
    Flight path
    Ground_track
    Heading
    Track

Icon

Interrogator

Mode

Separation minima

Surveillance area

Transponder

Structural Taxonomy

Advisory message

Aircraft
    Cockpit
    Flight characteristics

Air traffic control center

Collision warning situation
    Collision warning characteristics

Corrective actions advisory message

Icon

Inter_air message

Interrogator

Mode

Normal situation

Separation minima

Situation
    Relative_bearing
    Range
    Time_to_intersect
    Intersection

Surveillance area

Transponder

3. DOMAIN ASSUMPTIONS

COMMONALITY ASSUMPTIONS

This section lists the assumptions of commonality for the ATD/CWM domain. Justification is provided for each commonality.

The ATD/CWM assumptions of commonality are:

1. An ATD/CWM system maintains the notion of host aircraft.

    JUSTIFICATION: An ATD/CWM system is an embedded computer system installed in an aircraft to monitor air traffic in a surveillance area. A given ATD/CWM system detects collision warning situations relative to the aircraft on which it is installed. This aircraft is called the host aircraft.

2. An ATD/CWM system monitors a surveillance area.

> *JUSTIFICATION:* The purpose of an ATD/CWM system is to detect collision warning situations and take appropriate actions. To detect these situations, the ATD/CWM system must monitor air traffic in a given region called the surveillance area.

3. An ATD/CWM system monitors the potential threat flight characteristics ground_track, relative_bearing, range, altitude, airspeed, vertical rate of change, and aircraft_identification within the host aircraft's surveillance area.

> *JUSTIFICATION:* The flight characteristics ground_track, relative_bearing, range, and altitude determine an aircraft's location. The flight characteristics airspeed and vertical rate of change determine how the aircraft's location will change over time. The aircraft_identification is used to correlate information obtained at different times.

4. An ATD/CWM system predicts the aircraft flight path based on known ground_track, airspeed, and climb rate.

> *JUSTIFICATION:* The flight path of an aircraft must be determined so that the ATD/CWM can issue appropriate commands to avoid a collision. The flight path direction is determined by the aircraft's bearing. Predicting the aircraft's location on the flight path is determined by airspeed and vertical rate of change.

5. An ATD/CWM system monitors the probable intersection of all aircraft with the host aircraft to ensure that a separation minima is maintained.

> *JUSTIFICATION:* The separation minima is fixed by the FAA and probably will not change in the foreseeable future.

6. An ATD/CWM system detects collision warning situations with respect to each potential threat based on its predicted flight path and the separation minima.

> *JUSTIFICATION:* This is the purpose of an ATD/CWM system.

7. An ATD/CWM system must detect the occurrence of a collision warning situation within a prescribed time period.

> *JUSTIFICATION:* The safety of the host aircraft depends upon the ATD/CWM system detecting collision warning situations in a timely manner.

8. An ATD/CWM system takes some action for recognized collision warning situations.

> *JUSTIFICATION:* Once a collision warning situation has been detected, it is necessary to notify the host aircraft pilot so that he can take appropriate steps to avoid a collision.

9. An ATD/CWM system must perform the appropriate actions for recognized collision warning situations within a prescribed time period.

> *JUSTIFICATION:* Notifying the host aircraft within this time period allows the pilot time to avoid the collision.

10. An ATD/CWM system will exhibit a corrective action advisory message on the host aircraft's display when the system detects a collision warning situation. This message describes maneuvers the host aircraft should perform to avoid a collision. This message only occurs for specific collision warning situations.

> *JUSTIFICATION:* Since the pilot is responsible for flying the aircraft, he must be told what specific maneuvers to perform to avoid a collision.

11. An ATD/CWM system determines the minimal range separating a potential threat and host aircraft using their respective predicted flight paths.

> *JUSTIFICATION:* The minimal range separation determines the content of the corrective action advisory message.

12. An aircraft can progress through different collision warning situations relative to the host aircraft.

> *JUSTIFICATION:* All aircraft, including the host aircraft, have flight characteristics. Collision warning situations are based on these flight characteristics. As these characteristics change over time, a given aircraft could be in different collision warning situations, relative to the host aircraft, at different points in time.

13. No single aircraft can be in more than one collision warning situation, relative to the host aircraft, at any given time.

> *JUSTIFICATION:* It is useful for an ATD/CWM system to maintain the notion of differing severity levels of collision warning situations. The situations are disjointed to allow for partitioning of evasive actions based on the collision warning situation.

14. An ATD/CWM system has interfaces with radar and ATC devices that enable it to receive flight characteristic information on a potential threat.

> *JUSTIFICATION:* The ATD/CWM system must receive information on potential threats to detect collision warning situations. Onboard systems are insufficient for determining all necessary aircraft flight characteristics.

15. An ATD/CWM system has an interface with a navigation device that enables the ATD/CWM system to receive flight characteristic information on the host aircraft.

> *JUSTIFICATION:* An ATD/CWM system detects collision warning situations relative to the host aircraft. Host aircraft flight characteristics, which are necessary to detect these situations, are obtained from the onboard navigation device.

16. An ATD/CWM system has an interface to a display device (ATD) that enables the ATD/CWM system to represent aircraft, potential threat flight characteristics, and display advisory messages in the host aircraft's cockpit.

> *JUSTIFICATION:* The pilot needs to know what situations have been detected and the locations of the corresponding potential threats so that he can perform the appropriate evasive actions.

17.   An ATD/CWM system must execute within a fixed memory size.

      JUSTIFICATION: Since an ATD/CWM system is embedded within the host aircraft, there are physical size constraints which limit the amount of memory associated with the processing unit. Consequently, the ATD/CWM system must perform its functions within this limit.

18.   An ATD/CWM system performs CPU-intensive numerical calculations that require hardware support for floating point operations.

      JUSTIFICATION: It is doubtful that software-emulated floating point arithmetic will allow an ATD/CWM system to meet its performance requirements.

19.   An ATD/CWM system executes on a single processor hardware platform.

      JUSTIFICATION: Processing speeds of commercially available processors are sufficient to allow an ATD/CWM system to meet its requirements. Furthermore, there is substantial cost savings of a single processor system compared to a multi-processor configuration.

20.   An air traffic display supports at least ten colors, displays text, and provides capabilities for displaying graphical shapes and manipulating their color, location, and blinking attributes.

      JUSTIFICATION: The display is used to convey aircraft, their location, and collision warning situations between the host aircraft and potential threats. This type of information is easily conveyed to the pilot through the use of color, geometric shape, and blinking capabilities.

21.   An ATD/CWM system represents aircraft on the display as icons.

      JUSTIFICATION: Human factors studies have determined the pilot can easily recognize and understand aircraft displayed as icons.

22.   An audible alarm is characterized by a frequency, duration, and volume. The volume of the audible alarm is fixed.

      JUSTIFICATION: The quantities that characterize the **sound** made by an audible alarm are: how long it was rung (duration), how loud it is (volume), and the pitch (frequency). It is anticipated that audible alarm devices will not permit the volume of the alarm to be controlled by software.

23.   An ATD/CWM system receives range information on potential threats from the onboard radar device.

      JUSTIFICATION: Onboard radars currently found on aircraft provide range information. It is anticipated that all future radar devices will continue to provide at least this information.

24.   An ATD/CWM system receives the host aircraft altitude, airspeed, ground_track, and location from the onboard navigation device.

      JUSTIFICATION: Navigation devices currently provide this information and it is anticipated that all future navigation devices will continue to provide at least this information.

25. An ATD/CWM system receives range, altitude, airspeed, ground_track, and relative_bearing from the ATC device for aircraft.

> *JUSTIFICATION:* Air traffic control centers currently provide this information, and it is anticipated that this information will continue to be provided in the future.

VARIABILITY ASSUMPTIONS

This section lists the assumptions of variability for the ATD/CWM domain. Justification is provided for each variability.

The ATD/CWM assumptions of variability are:

1. The definitions and severity of collision warning situations recognized by the ATD/CWM system

> *JUSTIFICATION:* Different members of the ATD/CWM domain can have different (and possibly multiple) collision warning situations. There must be a way of defining what they are.

2. Actions performed by the ATD/CWM system for each collision warning situation

> *JUSTIFICATION:* It is anticipated that the set of actions performed for each collision warning situation will differ as a function of the probability of a collision occurring.

3. The flight characteristics displayed for each aircraft maintained by the ATD system

> *JUSTIFICATION:* The information displayed on the ATD is a function of determining what information is important to see and how much a human can comprehend.

4. Format used to display flight characteristics for each aircraft maintained by the ATD system

> *JUSTIFICATION:* Formatting information for easy comprehension by a human is subject to human factors issues.

5. The geometry of the surveillance area covered by the ATD/CWM system in the host aircraft

> *JUSTIFICATION:* A surveillance area has a cylindrical shape. The bounds can be changed depending upon the current flight mode (e.g., enroute, terminal, departure, and arrival). The size differs for these flight modes because the needs of the surveillance area coverage are different.

6. Format of the message received from the navigation device

> *JUSTIFICATION:* Hardware changes due to advances in technology or cost reduction may cause the navigation device to be replaced. Its replacement may have a different message format.

7. Format of the message received from the radar device

> *JUSTIFICATION:* Hardware changes due to advances in technology or cost reduction may cause the radar device to be replaced. Its replacement may have a different message format.

8. The computer system used including processing speed. primary memory size, and availability of secondary memory

   JUSTIFICATION: Technical advancements in hardware may improve processor speed. lower the cost of a processor, and lower the cost of memory. Consequently, the computer system configuration may change (e.g., more memory. faster central processing unit).

9. Format of inter_air messages transmitted to potential threats

   JUSTIFICATION: Hardware changes due to advances in technology or cost reduction may cause the communication device to be replaced. Its replacement may have a different message format.

10. Format of advisory messages transmitted to air traffic control centers

   JUSTIFICATION: Hardware changes due to advances in technology or cost reduction may cause the communication device to be replaced. Its replacement may have a different message format.

11. Format of messages containing flight characteristics received from air traffic control centers

   JUSTIFICATION: Hardware changes due to advances in technology or cost reduction may cause the navigation device to be replaced. Its replacement may have a different message format.

12. Type of radar, navigation, ATC, communication, audible alarm, or display device

   JUSTIFICATION: Advances in hardware technology may make current devices obsolete in terms of speed, capabilities, or cost. These advances may consequently cause replacement of a device with another one.

13. The maximum number of aircraft monitored by the ATD/CWM system at any given instant

   JUSTIFICATION: An ATD/CWM system must be able to handle a defined number of aircraft per time unit.

14. The presence of an audible alarm and the frequency and duration of the audible sound

   JUSTIFICATION: Ringing an audible alarm is one action an ATD/CWM system can perform to notify the host aircraft of a particular collision warning situation. However, ringing the audible alarm to signify different collision warning situations requires a distinct frequency and duration for each so that the pilot (or other flightcrew member) can uniquely identify the corresponding collision warning situation. Furthermore, the set of actions performed by an ATD/CWM system in response to a collision warning situation is variable. If **ringing the audible alarm** is not one of them, then the generated system does not need to include the audible alarm. There could be a resulting cost savings.

4. DOMAIN STATUS

The variabilities referenced in parentheses are listed in the Variabilities Assumptions section on pages 2-10 and 2-11.

- These variabilities are not accommodated by the remaining ATD/CWM domain engineering work products.

    - Maximum number of aircraft monitored at any given instant (Variability 13).

    - The flight characteristics displayed for each aircraft maintained by the ATD system (Variability 3).

    - Format of message received from the navigation device (Variability 6).

    - Format of message received from the radar device (Variability 7).

    - Computer system, including memory size, availability of secondary memory, processor (Variability 8).

    - Format of inter_air message transmitted to potential threats (Variability 9).

    - Format of messages received from air traffic control centers (Variability 11).

    - Type of devices for radar, navigation, ATC, communication, audible alarm, and display (Variability 12).

- These variabilities are accommodated in a restricted manner by the remaining ATD/CWM domain engineering work products.

    - Format used to display flight characteristics for each aircraft maintained by the ATD system (Variability 4).

    - Surveillance area geometry (Variability 5).

    - Format of advisory message transmitted to the air traffic control center (Variability 10).

3. ATD/CWM DECISION MODEL

NOTE: The Decision Model is written in a mechanically processable syntax shown below. In this form, mnemonics for the decisions (i.e., requirements variations) are introduced because the decisions are needed in writing the adaptable product requirements. The description of a decision in the Decision Model is characterized by the following information:

- Mnemonic – Name for the decision.

- Repetition factor – An optional symbol which is one of "?" (zero or one), " + " (one or more), or "*" (zero or more). This indicates how many times the decision represented by the mnemonic can be repeated.

- Description – An optional textual description delimited by enclosing parentheses.

- Domain – Describes the value space for the mnemonic. The value space can be simple (e.g., integer, identifier, natural) or composite. A composite value space is indicated by the keywords "and" or "or" followed by another decision. The keyword "and" means that all of the decisions must be made for the composite. The keyword "or" means that only one of the following decisions can be made for the composite for a given instance of the decision group.

The following BNF notation is used to describe the decision model.

DECISION ::= MNEMONIC [" + " | "*" | "?"] [(DESCRIPTION)] ":" DOMAIN

DOMAIN ::=   SIMPLE_DOMAIN
              | "and" DECISION_SET
              | "or" DECISION_SET

DECISION_SET ::= DECISION [DECISION_SET]

NOTE: Definitions for terms bracketed by exclamation points (e.g., !xxx!) are found on page 4-21.

DECISION GROUPS AND SPECIFICATION

The Decision Model consists of the following groups.

Project_Information

The Project_Information (PI) describes the specific information for the project. The decisions that must be made for this decision class are:

Project_Information : and

Contract   (Contract information for the specific project.) : and

                 Agency   (Name of the contracting agency.) : identifier(1..64)

                 Number   (Contract number (can be an alphanumeric thereby excluding a
                           numeric value space) : identifier(1..64)

                 CDRL   (Contract data requirements list number.) : identifier(1..64)

System   (System information for the specific project.) : and

                 Name   (System name.) : identifier(1..64)

                 Mnemonic (System mnemonic.) : identifier(1..16)

                 Id   (System identification number.) : identifier(1..64)

Surveillance_Area

The Surveillance_Area (SA) is the area around an aircraft that an ATD/CWM system monitors. The
decision that must be made for this decision class is:

Surveillance_Area : and

Range   (Radius, in nautical_miles, of the surveillance area around an aircraft that is
                 monitored by the ATD/CWM system. The surveillance area is a sphere whose
                 origin is the host aircraft's position.) : nautical_miles(10..300)

NOTE: To simplify the domain implementation, the geometry of the surveillance area is assumed to
be spherical. The radius of this sphere is fixed at system generation time. It is assumed that
the onboard radar device has a range of at least 300 nautical_miles.

Collision_Warning_Situation

The Collision_Warning_Situation (CWS) is a situation that the ATD/CWM system detects between
a potential threat and the host aircraft. A given potential threat can progress through different colli-
sion warning situations relative to the host aircraft as the potential threat navigates through the host
aircraft's surveillance area. The decisions that must be made for this decision class are:

Collision_Warning_Situation + : and

CWS_Name (Name of this collision warning situation. There is a predefined instance of this
           class called **normal**.) : identifier(1..64)

CWS_Def   (Boolean-valued expression that defines the criteria for this collision warning
           situation. Either Time, Range, or both can be specified. However, at least one
           of these must be specified. If both are specified, then the expression is
           interpreted as a logical OR of both components [i.e., Time OR Range]) : or

Time   (Time is the time_to_intersect measured in seconds with the potential threat maintaining its current airspeed, course, and climb rate. If the airspeed is unknown, then 1,000 nautical_miles per hour is assumed. The time_to_intersect lies in the range Time.Min $\leq$ time_to_intersect $<$ Time.Max) : and

    Min   (Minimum allowed elapsed time before the flight path's of the potential threat and host aircraft intersect.) : seconds(1..300)

    Max   (Upper bound on the allowed elapsed time before the flight path's of the potential threat and host aircraft intersect.) : seconds (1..300)

Range   (Range is the distance the potential threat is from the host aircraft measured in nautical_miles. The range value lies in the range Range.Min $\leq$ range $<$ Range.Max) : and

    Min   (Minimum distance the potential_threat is from the host aircraft. The upper limit is determined by the Surveillance_Area range.) : nautical_miles(0.. < SA.Range >)

    Max   (Upper bound on the distance the potential threat is from the host aircraft. The upper limit is determined by the Surveillance_Area range.) : nautical_miles(0.. < SA.Range >)

Partition   (Indicates the potential threat partition for which this collision warning situation applies. ID is **identified**; UID is **unidentified**; ALL is both.) : enumerated (ID, UID, ALL)

Severity   (Relative probability that a collision is likely to occur. The higher the severity is, the more likely a collision will occur. By definition, the predefined normal situation has the lowest severity.) : severity(0.00 .. 1.00, 0.01)

Response   (Prescribed response to the collision warning situation.) : CWSR

Collision_Warning_Situation_Response

The Collision_Warning_Situation_Response (CWSR) is the actions that an ATD/CWM system can perform in response to a detected collision warning situation. The decisions that must be made for this decision class are:

Collision_Warning_Situation_Response+ : and

    ATC_Msg   (Designates whether a message is sent to the nearest air traffic control center. "true" means to send the message.) : enumerated (true, false)

    Inter_Air_Msg(Designates whether a message is sent to the appropriate potential threat. "true" means to send the message.) : enumerated (true, false)

Corrective_Msg (Designates whether a corrective action advisory message is displayed on the host aircraft's display. "true" means to send the message.) : enumerated (true, false)

Alarm   (Designates whether the audible alarm should be rung when a potential threat migrates into this collision warning situation from a lower priority situation.) : or

    No_Ring   (Do not ring the alarm.) : true

    Ring   (Ring the alarm. In this case, a frequency and duration must be specified as well.) : and

        Pitch   (What frequency, in hertz, to ring the audible alarm.) : hertz(1,000..10,000)

        Duration   (How long, in seconds, to ring the audible alarm.) : seconds(0.01 .. 10.0, 0.01)

Code   (Transponder code used in the message sent to the appropriate potential threat [Inter_Air_Msg] air traffic control center [ATC_Msg].) : !transponder_code!

ATC_Message

The ATC_Message describes the message format used when sending messages from the host aircraft to an air traffic control center. The decision that must be made for this decision class is:

ATC_Message : and

Mode   (Designates the format for ATC_Msg messages sent from the host aircraft to an air traffic control center. Mode A is a 4-digit !transponder_code!: Mode C is a 4-digit !transponder_code! and host aircraft altitude.) : enumerated (A, C)

Aircraft_Status_Display

The Aircraft_Status_Display (ASD) describes how to display the information status for a potential threat when the potential threat is in a specific aircraft partition and collision warning situation relative to the host aircraft. The decisions that must be made for this decision class are:

Aircraft_Status_Display + : and

    Situation   (Indicated collision warning situation.) : CWS

    Partition   (Indicated aircraft partition. ID is **identified**; UID is **unidentified**.) : enumerated (ID, UID)

    PT_Color   (Icon color for the potential threat.) : enumerated (red, orange, green, yellow, white, blue, black, pink, purple, indigo, violet)

    PT_Blink   (Designates whether the potential threat icon should blink for this collision warning situation. True means blink, false means do not blink.) : enumerated (true, false)

PT_Fill　　　　(Icon filling for the potential threat involved in this collision warning situation. True means to fill the icon [i.e., color the icon interior]; false means do not fill in the icon.) : enumerated (true, false)

NOTE: The icon interior is colored the same as the icon color.

Host_Aircraft_Status_Display

The Host_Aircraft_Status_Display (HASD) describes how to display the information status for the host aircraft when it is involved in a specific collision warning situation. The decisions that must be made for this decision class are:

Host_Aircraft_Status_Display+ : and

　　　Situation　　　　(Indicated collision warning situation.) : CWS

　　　Color　　　　(Icon color for the host aircraft.) : enumerated (red, orange, green, yellow, white, blue, black, pink, purple, indigo, violet)

NOTE: The host aircraft color is independent of whether the potential threat is **identified** or **unidentified**.

Aircraft_Display_Symbol

The Aircraft_Display_Symbol (ADS) describes the two-dimensional icon shape for displaying aircraft on the ATD. The decisions that must be made for this decision class are:

Aircraft_Display_Symbol : and

　　　Host_Shape　　(Icon shape for the host aircraft.) : enumerated (circle, square, triangle)

　　　ID_Shape　　　(Icon shape for **identified** potential threats.) : and

　　　　　　Partition　　(Boolean-valued expression that defines the criteria for an **identified** potential threat.) : !id_definition!

　　　　　　Shape　　　(Icon shape for **identified** potential threats.) : enumerated (circle, square, triangle)

　　　UID_Shape　　(Icon shape for **unidentified** potential threats.) : enumerated (circle, square, triangle)

NOTE: A potential threat whose flight characteristics do not satisfy the !id_definition! is considered to be unidentified. The icon shape is solely dependent upon whether the aircraft is the host aircraft, an identified potential threat, or unidentified potential threat.

DEPENDENCY CONSTRAINTS

Table 3-1 shows the dependency constraints for decision classes in the ATD/CWM domain.

Table 3-1. Dependency Constraints

Decision Class	Condition	Dependency Constraint
CWSR	True	At least one Collision_Warning_Situation_Response must have the Corrective_Msg decision **true**.
ATC_Format	True	A decision must be made for the ATC_Msg message format only when at least one Collision_Warning_Situation_Response has the ATC_Msg decision **true**.
CWS	Range.Min $\leq$ < SA.Range > Range.Max $\leq$ < SA.Range >	The minimum or maximum distance for a collision warning situation definition cannot be larger than the range specified for the surveillance area.
CWS	Range.Min $\leq$ Range.Max	The minimum range value must be less than or equal to the maximum range value for a given collision warning situation.
CWS	Time.Min $\leq$ Time.Max	The minimum time value must be less than or equal to the maximum time value for a given collision warning situation.
CWS	Time Range	Either Time, Range, or both must be specified for a collision warning situation definition.
ADS	Host_Shape ID_Shape.Shape UID_Shape	The symbols for the icon shape must be mutually exclusive.

4. ATD/CWM PRODUCT REQUIREMENTS

NOTE: The requirements method used to capture the ATD/CWM Product Requirements is described in Appendix A.

1. THEORY

This section defines a model of the underlying theory of the ATD/CWM system. This model provides the basis for the description of system behavior in Section 3. The components of the model are concept/entity classes and derivation relations. Entities in the model correspond to real world entities that are of concern to the ATD/CWM application.

1.1. STATIC MODEL

This section describes the classes of entities that model the information about aircraft and collision warning situations embedded in ATD/CWM applications.

1.1.1. Aircraft

Each aircraft has the following properties.

Property	Type	Description
Altitude	feet(-200 .. 50,000)	Aircraft altitude measured in feet.
Aircraft_ID	string(8)	Aircraft identifier.
Velocity	speed(1 .. 1,500)	Aircraft airspeed measured in knots.
Climb Rate	rate(-3,000 ... 3,000)	Aircraft's rate of change in altitude measured in feet per minute.
Location	!location!	Latitude and longitude location of the aircraft given as a tuple (latitude, longitude). Latitude has a permitted range of -90.0 ... 90.0 degrees. A negative value indicates latitude south of the equator; a positive value is north of the equator. Longitude has a permitted range of -360.0 ... 360.0 degrees. A positive value signifies longitude west of the prime meridian at Greenwich, England. A negative value indicates longitude east of the prime meridian. Resolution for latitude and longitude is one-tenth degree.
Ground_Track	degrees	Ground_track is measured from the line of the aircraft to magnetic north to the horizontal component of the aircraft's actual flight path over the surface of the earth. Ground_track is measured in degrees with a resolution of one degree.

1.1.2. Host_Aircraft : Aircraft

Host Aircraft is a singleton class representing the aircraft on which the ATD/CWM system is installed.

1.1.3. Potential_Threat : Aircraft

Potential_Threat is a class of aircraft that represents all aircraft within the host aircraft's surveillance area (except the host aircraft itself). Potential threats have the following additional properties.

Property	Domain	Description
Range	nautical miles	Range of the potential threat from the host aircraft. Range is measured in nautical_miles with a resolution of one nautical_mile.
Relative_Bearing	degrees	Bearing of the potential threat relative to the host aircraft. Relative_bearing is measured from the ground_track of the host aircraft to the line from the host aircraft to the potential threat in the clockwise direction looking down.

1.2. Dynamic Model

1.2.1. Time to Intersect

The ATD/CWM system determines how much time elapses (time_to_intersect) until the flight paths of the host aircraft and potential threat cross within a separation minima. The separation minima is the minimal range that the two aircraft will be apart assuming constant velocity, ground_track, and climb rate for both aircraft. The following equation gives range between the host aircraft and a potential threat at a given point in time as a function of their respective locations in space. The terms (x_H, y_H, z_H) and (z_{PT}, y_{PT}, z_{PT}) are the rectangular coordinates of the host aircraft (H) and potential threat (PT), respectively.

$$\text{range} = \sqrt{(x_{PT} - x_H)^2 + (y_{PT} - y_H)^2 + (z_{PT} - z_H)^2}$$

The following equations give the location of the host aircraft over time (assuming a constant velocity, ground_track, and climb rate) where (x_{oH}, y_{oH}, z_{oH}) denote an initial location of the host aircraft.

$$x_H = x_{oH} + \text{host\_aircraft.velocity}_{XY} * \cos(\text{host\_aircraft.ground\_track}) * t$$

$$y_H = y_{oH} + \text{host\_aircraft.velocity}_{XY} * \sin(\text{host\_aircraft.ground\_track}) * t$$

$$z_H = z_{oH} + \text{host\_aircraft.climb\_rate} * t$$

Similarly, the following equations give the location of the potential threat (assuming a constant velocity, ground_track, and climb rate) where $(x_{oPT}, y_{oPT}, z_{oPT})$ denote an initial location of the potential threat.

$$x_{PT} = x_{oPT} + \text{potential\_threat.velocity}_{XY} * \cos(\text{potential\_threat.ground\_track}) * t$$

$$y_{PT} = y_{oPT} + \text{potential\_threat.velocity}_{XY} * \sin(\text{potential\_threat.ground\_track}) * t$$

$$z_{PT} = z_{oPT} + \text{potential\_threat.climb\_rate} * t$$

The potential threat location is always relative to the host aircraft. By assuming that the host aircraft location gives the origin of the rectangular coordinate system, the following equations give an initial location of the potential threat relative to the host aircraft.

$$x_{oPT} = \text{range}_{XY} * \cos(\text{potential\_threat.relative\_bearing})$$

$$y_{oPT} = \text{range}_{XY} * \sin(\text{potential\_threat.relative\_bearing})$$

$$z_{oPT} = \text{potential\_threat.altitude} - \text{host\_aircraft.altitude}$$

The following equation gives the horizontal component of the velocity (i.e., the component of velocity that lies in the X-Y plane - velocity_{XY}).

$$\text{velocity}_{XY} = \sqrt{\text{velocity}^2 - \text{climb\_rate}^2}$$

Similarly, the following equation gives the horizontal component of the range (i.e., the component of range that lies in the X-Y plane - range_{XY}).

$$\text{range}_{XY} = \sqrt{\text{range}^2 - (\text{host\_aircraft.altitude} - \text{potential\_threat.altitude})^2}$$

By making appropriate substitutions, the range between the host aircraft and a potential threat is a function of their initial locations and time.

$$\text{range} = f(t, x_{oH}, y_{oH}, z_{oH}, x_{oPT}, y_{oPT}, z_{oPT})$$

The **time_to_intersect** is computed by taking the first derivative of the range equation expressed as a function of time, setting it equal to zero, and solving for "t" (assuming (x_{oH}, y_{oH}, z_{oH}) is $(0,0,0)$).

$$f'(t, x_{oH}, y_{oH}, z_{oH}, x_{oPT}, y_{oPT}, z_{oPT}) = 0$$

This yields a **time_to_intersect** value - t_{min}.

1.2.2. Minimal Separation Distance

The ATD/CWM system determines the minimal separation distance (i.e., minimal range—the closest range of these two aircraft with respect to each other) between the host aircraft and potential threat assuming constant velocity, ground_track, and climb rate for each. This minimal range (range_{min}) is determined by solving the following equation with $t = t_{min}$ (assuming (x_{oH}, y_{oH}, z_{oH}) is $(0,0,0)$).

$$\text{range}_{min} = f(t, x_{oH}, y_{oH}, z_{oH}, x_{oPT}, y_{oPT}, z_{oPT})$$

1.2.3. Climb Rate

The following equation is used to compute the climb_rate of an aircraft.

$$climb\_rate = \left(\frac{aircraft.altitude_{t_b} - aircraft.altitude_{t_a}}{t_b - t_a} \right)$$

Terms t_b and t_a denote time and $t_b > t_a$. At system startup ($t_a = 0$), altitude = 0. 5000 feet per minute (fpm) is used if the climb rate exceeds 5000 fpm.

1.2.4. Relative Bearing

The ATD/CWM system obtains range for a potential threat from two sources: the radar and ATC. When the radar range for a specific potential threat is newer than the range obtained in the ATC_Message for the same potential threat, it is necessary to predict the location of the potential threat. The location is computed in terms of its relative_bearing based on the time difference between the most recent radar range and ATC range values assuming constant velocity, climb rate, and ground_track for both the host aircraft and potential threat. In the following discussion, t_a and t_b denote time and $t_b > t_a$.

The following equations give the x and y rectangular coordinates of the potential threat at t_b (x_{PT}, y_{PT}) where (x_{oPT}, y_{oPT}) is its x and y coordinates at time t_a.

$$x_{nPT} = x_{oPT} + potential\_threat.velocity_{XY} * cos(potential\_threat.ground\_track) * (t_b - t_a)$$

$$y_{nPT} = y_{oPT} + potential\_threat.velocity_{XY} * sin(potential\_threat.ground\_track) * (t_b - t_a)$$

Similarly, the following equation gives the x rectangular coordinate of the host aircraft at time t_b (assuming that the initial location x_{oH} is 0).

$$x_{nH} = host\_aircraft.velocity_{XY} * cos(host\_aircraft.ground\_track) * (t_b - t_a)$$

The following equations give the initial x and y rectangular coordinates at time t_a (x_{oPT}, y_{oPT}) assuming that the host aircraft gives the origin of the rectangular coordinate system.

$$x_{oPT} = range_{XY} * cos(potential\_threat.relative\_bearing)$$

$$y_{oPT} = range_{XY} * sin(potential\_threat.relative\_bearing)$$

Since the range, relative_bearing, velocity, ground_track, and climb rate are known for time t_a, a value for x_{nPT} can be computed. The new relative bearing of the potential threat is a function of the new x_{nPT}, the new location of the host aircraft x_{nH}, and the new range as shown below.

$$x_{nPT} = range_{XY} * cos(potential\_threat.relative\_bearing) + x_{nH}$$

Rewriting the equation to solve for relative_bearing yields the following equation.

$$potential\_threat.relative\_bearing = arccos\left(\frac{x_{nPT} - x_{nH}}{range_{XY}} \right)$$

The horizontal component of the range at time t_b is given by the following equation.

$$\text{range}_{XY} = \sqrt{\text{range}^2 - \left(\text{host\_aircraft.altitude}_{t_b} - \text{potential\_threat.altitude}_{t_b}\right)^2}$$

The altitude of the host aircraft and potential threat at time t_b is predicted using their respective altitudes at time t_a assuming a constant climb rate as shown below.

$$\text{host\_aircraft.altitude}_{t_b} = \text{host\_aircraft.altitude}_{t_a} + (t_b - t_a) * \text{host\_aircraft.climb\_rate}$$

$$\text{potential\_threat.altitude}_{t_b} = \text{potential\_threat.altitude}_{t_a} + (t_b - t_a) * \text{potential\_threat.climb\_rate}$$

The sign of the y rectangular coordinate at time t_b (y_{nH}) is used to determine whether the potential threat relative bearing must be adjusted. If the sign of y is negative, then

$$\text{potential\_threat.relative\_bearing} = 360 - \text{potential\_threat.relative\_bearing}$$

1.2.5. Potential Threat Collision Warning Situation Status

The ATD/CWM system monitors potential threats within the host aircraft's surveillance area and determines when they make the transition from one collision warning situation to another. The collision warning situation the potential threat is in, relative to the host aircraft, determines the collision warning situation status (cws_status) of a potential threat as follows. The following abbreviations are used in this algorithm.

   pt_R potential_threat.range
   pt_T time_to_intersect between the potential threat and host aircraft.

```
< forall C in Collision_Warning_Situation >
    < if C.CWS_Def = {Time, Range} then >
        @T(   (< C.CWS_Def.Time.Min > ≤ ptT < < C.CWS_Def.Time.Max > ) OR
              (< C.CWS_Def.Range.Min > ≤ ptR < < C.CWS_Def.Range.Max > ))
                    when partition = < C.Partition >
              cws_status = < C.CWS_Name >
    < else if C.CWS_Def = {Time} then >
        @T(< C.CWS_Def.Time.Min > ≤ ptT < < C.CWS_Def.Time.Max > )
                    when partition = < C.Partition >
              cws_status = < C.CWS_Name >
    < else if C.CWS_Def = {Range} then >
        @T(< C.CWS_Def.Range.Min > ≤ ptR < < C.CWS_Def.Range.Max > )
                    when partition = < C.Partition >
              cws_status = < C.CWS_Name >
    < endif >
< endfor >
@T(ptR < < Surveillance_Area.Range > ) when partition = ALL
              cws_status = Normal
```

If more than one condition is true for a given potential threat, then the condition having the highest severity level gives the collision warning situation status (cws_status).

1.2.6. Potential Threat Partition

The ATD/CWM system monitors potential threats within the host aircraft's surveillance area to determine when they make the transition from being identified to unidentified and vice-versa. The partition of a potential threat is defined as follows:

```
if <Aircraft_Display_Symbol.ID_Shape.Partition>
    partition = identified
else
    partition = unidentified
endif
```

1.2.7. Host Aircraft Collision Warning Situation Status

The potential threats in the host aircraft's surveillance area determine the host aircraft's collision warning situation status (cws_status). The host aircraft's cws_status is equal to the potential threat cws_status that has the highest severity from all known potential threats. The host aircraft's icon color on the air traffic display will display the following colors for the corresponding collision warning situation status:

Collision Warning Situation Status	Color
<forall S in Host_Aircraft_Status_Display>	
<Host_Aircraft_Status_Display.Situation>	<Host_Aircraft_Status_Display.Color>
<endfor>	
Normal	White

2. ENVIRONMENT

This section describes the external environment within which the ATD/CWM system operates.

2.1. INPUT COMMUNICATION PATHS

2.1.1. Navigation

The navigation device communicates host aircraft flight characteristics (i.e., altitude, velocity, ground_track, latitude, and longitude) to the host aircraft.

Input Data Item: Navigation_Msg (5 data items)

Name	Value Space	Description / Data Representation
altitude	Unit: feet Range: 0 to 60,000 Accuracy: 1 foot Resolution: 1 foot	Vertical distance height of the host_aircraft measured from mean sea level. 32-bit natural number scale: 1 offset: 0
velocity	Unit: knots Range: 0 to 700 Accuracy: 1 knot Resolution: 1 knot	Indicated velocity of the host_aircraft. 16-bit natural number scale: 1 offset: 0

ground_track	Unit: degrees Range: 0 to 360 Accuracy: 0.1 degree Resolution: 0.1 degree	Ground_track is measured from the line of the aircraft to magnetic north to the horizontal component of the aircraft's actual flight path over the surface of the earth. 16-bit unsigned number scale: 10 offset: 0
latitude	Unit: degrees Range: -90 to 90 Accuracy: 0.1 degree Resolution: 0.1 degree	Latitude component of the location. Negative values represent latitude south of the equator. 16-bit two's complement number scale: 10 offset: 0
longitude	Unit: degrees Range: -360 to 360 Accuracy: 0.1 degree Resolution: 0.1 degree	Longitude component of the location. A positive value signifies longitude west of the prime meridian at Greenwich, England. A negative value indicates longitude east of the prime meridian. 16-bit two's complement number scale: 10 offset: 0

Timing Characteristics: TBD seconds

Input Mapping: context: (host_aircraft, true)
host_aircraft.altitude = altitude
host_aircraft.velocity = velocity
host_aircraft.ground_track = ground_track
host_aircraft.location = (latitude, longitude)

2.1.2. Radar

The radar device provides information (i.e., flight characteristics) about external aircraft. The radar sweep rate is 0.25 seconds.

Input Data Item: Radar_Msg (4 data items)

Name	Value Space	Description / Data Representation
aircraft_id	string(8)	Aircraft identification. 64-bit string
sweep	Unit: sweep_count Range: modulo 32 Accuracy: 1 Resolution: 1	Radar sweep number modulo 32. 5-bit positive number scale: 1 offset: 0

range	Unit: nautical miles Range: 0 to 300 Accuracy: 0.1 nautical mile Resolution: 0.00001525 nautical mile	Distance in nautical miles from this aircraft to the host_aircraft. 27-bit natural number scale: 65.536 offse.. 0
relative_ bearing	Unit: degrees Range: 0 to 360 Accuracy: 0.1 degree Resolution: 0.1 degree	Bearing to this aircraft relative to the host_aircraft. 16-bit unsigned number scale: 10 offset: 0

Timing Characteristics: TBD

Input Mapping: context: (potential_threat, potential_threat.aircraft_id = aircraft_id)
potential_threat.range = range
potential_threat.relative_bearing = relative_bearing

2.1.3. ATC

The ATC device provides information (i.e.. flight characteristics) about external aircraft. The device interrupts the ATD/CWM system for every message received. The message received from this device has the following components.

Input Data Item: ATC_Message

Name	Value Space	Description / Data Representation
aircraft_id	string(8)	Aircraft identification. 64-bit string
altitude	Unit: feet Range: 0 to 60,000 Accuracy: 1 foot Resolution: 1 foot	Vertical distance height of the potential threat measured from mean sea level. 32-bit natural number scale: 1 offset: 0
velocity	Unit: knots Range: 0 to 700 Accuracy: 1 knot Resolution: 1 knot	Indicated airspeed of the host aircraft. 16-bit natural number scale: 1 offset: 0
relative_ bearing	Unit: degrees Range: 0 to 360 Accuracy: 0.1 degree Resolution: 0.1 degree	Bearing of the potential threat relative to the host aircraft. Relative_bearing is measured from the ground_track of the host aircraft to the line from the host aircraft to the potential threat in the clockwise direction looking down. 16-bit unsigned number scale: 10 offset: 0

range	Unit: nautical miles Range: 0 to 300 Accuracy: 0.1 nautical mile Resolution: 0.1 nautical mile	Distance in nautical miles from this aircraft to the host aircraft. 16-bit unsigned number scale: 10 offset: 0
ground_track	Unit: degrees Range: 0 to 360 Accuracy: 0.1 degree Resolution: 0.1 degree	Ground_track is measured from the line of the aircraft to magnetic north to the horizontal component of the aircraft's actual flight path over the surface of the earth. 16-bit unsigned number scale: 10 offset: 0
timestamp	Range: 0000 .. 2400 The leftmost two digits are the hours; the rightmost two digits are the minutes.	Timestamp of when the data was valid. The timestamp is four digits representing the hours and minutes from the 24-hour clock. 16-bit unsigned number

Timing Characteristics: TBD

Input Mapping: context: (potential threat, potential threat.aircraft_id = aircraft_id)
potential threat.altitude = altitude
potential threat.velocity = velocity
potential threat.ground_track = ground_track
potential threat.relative_bearing = relative_bearing
potential threat.range = range

2.2. OUTPUT COMMUNICATION PATHS

< if there exists A ∈ CWS such that A.Response.Alarm then >

2.2.1. Audible Alarm

The audible alarm is an audio cockpit signal characterized by frequency, duration, and a fixed volume level.

Output Data Item: Audible_Alarm_Msg (2 data items)

Name	Value Space	Description / Data Representation
frequency	Unit: hertz Range: 1,000 to 10,000 Accuracy: 1 hertz Resolution: 1 hertz	Pitch of the audible_alarm in hertz. 16-bit positive number scale: 1 offset: 0
duration	Unit: seconds Range: 0.01 to 10.0 Accuracy: 0.01 seconds Resolution: 0.01 seconds	How long to ring the audible_alarm. 16-bit unsigned number scale: 100 offset: 0

< endif >

2.2.2. ATD

The ATD is a color display accessible by the ATD/CWM system. The ATD manipulates pixels on a bitmap display and manipulates icon color, shape, shade, and blink characteristics. This display can also position, move, or delete an icon and display text. The upper left-hand corner is pixel (0,0). The x axis is across the display to the right; the y axis is down the display towards the bottom.

Output Data Item: ATD_Msg

Name	Value Space	Description / Data Representation
id	Virtual memory address.	Handle for the displayed object. 16-bit natural number scale: 1 offset: 0
shape	Value Encoding: square: (1) circle: (2) triangle: (3)	Icon shape. 16-bit positive number.
size	Range: 1 .. 1,000	Size in pixels of the icon. 16-bit positive number scale: 1 offset: 0
fill	Value_Encoding: none: (1) yellow: (2) pink: (3) orange: (4) red: (5) green: (6) blue: (7) indigo: (8) purple: (9) violet: (10) black: (11) white: (12)	Color for the icon interior. 16-bit positive number.
color	Value_Encoding: none: (1) yellow: (2) pink: (3) orange: (4) red: (5) green: (6) blue: (7) indigo: (8) purple: (9) violet: (10) black: (11) white: (12)	Color for the icon. 16-bit positive number.

Name	Value Space	Description / Data Representation
fill_blink_rate	Unit: seconds Range: 0.0 .. 10.0 Accuracy: 0.1 second Resolution: 0.1 second	Blinking rate for the icon interior. 16-bit unsigned number scale: 10 offset: 0
obj_blink_rate	Unit: seconds Range: 0.0 .. 10.0 Accuracy: 0.1 second Resolution: 0.1 second	Blinking rate for the icon. 16-bit unsigned number scale: 10 offset: 0
x_location	Range: 0 .. 1100	Horizontal pixel location for icon center. 16-bit natural number scale: 1 offset: 0
y_location	Range: 0 .. 1100	Vertical pixel location for the icon center. 16-bit natural number scale: 1 offset: 0

Output Data Item:Corrective_Action_Msg

Name	Value Space	Description / Data Representation
text	string(N)	A variable length message describing what actions the pilot should perform to avoid a potential collision. "N" is the number of characters in the message. 8*N-bit string
x_location	Range: 0 .. 1,100	Horizontal pixel location for the first character of the text. 16-bit natural number scale: 1 offset: 0
y_location	Range: 0 .. 1,100	Vertical pixel location for the first character of the text. 16-bit natural number scale: 1 offset: 0

< :f there exists M ∈ CWS such that
(M.Response.ATC_Msg OR M.Response.Inter_Air_Msg) then >

2.2.3. Communication

This device can send messages to either the nearest air traffic control center or to a specific potential threat.

< if there exists M ∈ CWS such that M.Response.ATC_Msg then >

Output Data Item:ATC_Msg

Name	Value Space	Description / Data Representation
destination	Value Encoding: 1	Destination code. 16-bit positive number
code	Range: 0000 to 7777 with each digit only having the range 0 .. 7.	The !transponder_code! indicating the specific collision warning situation the host_aircraft is in. 16-bit natural number scale: 1 offset: 0

< if ATC_Message.Mode = C then >

altitude	Unit: feet Range: 0 to 60.000 Accuracy: 1 foot Resolution: 1 foot	Vertical distance height of the host_aircraft measured from mean sea level. 32-bit natural number scale: 1 offset: 0

< endif >
< endif >

< if there exists M ∈ CWS such that M.Response.Inter_Air_Msg then >

Output Data Item:Inter_Air_Msg

Name	Value Space	Description / Data Representation
destination	Value Encoding: 0	Destination code. 16-bit positive number
code	Range: 0000 to 7777 with each digit only having the range 0 .. 7.	The !transponder_code! indicating the specific collision warning situation the host_aircraft is in. 16-bit natural number scale: 1 offset: 0
altitude	Unit: feet Range: 0 to 60.000 Accuracy: 1 foot Resolution: 1 foot	Vertical distance height of the host_aircraft measured from mean sea level. 32-bit natural number scale: 1 offset: 0

latitude	Unit: degrees Range: -90 to 90 Accuracy: 0.1 degree Resolution: 0.1 degree	Latitude component of the location. Negative values represent latitude south of the equator. 16-bit two's complement number scale: 10 offset: 0
longitude	Unit: degrees Range: -360 to 360 Accuracy: 0.1 degree Resolution: 0.1 degree	Longitude component of the location. A positive value signifies longitude west of the prime meridian at Greenwich, England. A negative value indicates longitude east of the prime meridian. 16-bit two's complement number scale: 10 offset: 0

    <endif>
<endif>

3. EXTERNAL BEHAVIOR

3.1. PRESENTATION

<if there exists A ∈ CWS such that A.Response.Alarm then>

3.1.1. Audible Alarm

    Paradigm: Audible_Alarm
        Context: potential_threat
        Pitch_and_Duration: (
               <forall A in Collision_Warning_Situation>
                 <if A.Response.Alarm then>
                   (<A.CWS_Def>, <A.Response.Alarm.Pitch>,
                                    <A.Response.Alarm.Duration>)
               <endif>
             <endfor>)

<endif>

<if there exists M ∈ CWS such that
                   (M.Response.ATC_Msg OR M.Response.Inter_Air_Msg) then>

3.1.2. Communication

    <if there exists M ∈ CWS such that M.Response.ATC_Msg then>

3.1.2.1. ATC_Msg

    Paradigm: Binary
        Context: potential_threat
        Template: (

```
                    < forall M in Collision_Warning_Situation >
                        < if M.Response.ATC_Msg then >
                            < if ATC_Message.Mode = A then >
                                ( < M.CWS_Def >. " < M.Response.Code > ")
                            < endif >
                            < if ATC_Message.Mode = C then >
                                ( < M.CWS_Def >, " < M.Response.Code > @host_aircraft.altitude")
                            < endif >
                        < endif >
                    < endfor >
        )

    < endif >

    < if there exists M ∈ CWS such that M.Response.Inter_Air_Msg then >
```

3.1.2.2. Inter_Air_Msg

```
    Paradigm: binary
        Context:   potential_threat
        Template:  (
            < forall M in Collision_Warning_Situation >
                < if M.Inter_Air_Msg then >
                    ( < M.CWS_Def >,   " < M.Response.Code > @host_aircraft.altitude@host_air-
craft.latitude@host_aircraft.longitude")
                < endif >
            < endfor >
        )

    < endif >
< endif >
```

3.1.3. ATD

3.1.3.1. ATD_Msg

```
    Paradigm: map
        Context: (potential threat, potential threat.range ≤ < Surveillance_Area.Range > )
        Position_Attribute: (potential threat.relative bearing, potential threat.range)
        Focus: (host aircraft, true)
        Image:
            < forall instance S in Aircraft_Status_Display >
                ( < S.situation.CWS_Def AND S.Partition >,
                    (   Shape: < S.Partition.Shape >
                        Color: < S.PT_Color >
                        Blink: < S.PT_Blink >
                        Fill: < S.PT_Fill >
                    )
                )
```

< endfor >

Labels: aircraft.altitude, aircraft.velocity, aircraft.aircraft_id

Coordinate_System: (2 * **< Surveillance_Area.Range >**, 2 * **< Surveillance_Area.Range >**)

3.1.3.2. Corrective_Action_Msg

The following table defines the content of the corrective_action message based upon initial conditions (given in the topmost row of the table) and conditions that hold when the host aircraft and potential threat are closest together (leftmost column). The following abbreviations are used in the table.

alt_H host aircraft.altitude

alt_{PT} potential threat.altitude

$rate_H$ host aircraft.climb_rate

$rate_{PT}$ potential threat.climb_rate

Paradigm: text

Context: potential threat

Template: (

	$alt_H \geq alt_{PT}$ AND $alt_H - alt_{PT} \geq 500$ feet	$alt_H \geq alt_{PT}$ AND $alt_H - alt_{PT} < 500$ feet
msd $\geq$ 500 feet	maintain current heading and rate	maintain current heading and rate
msd < 500 feet AND $rate_H = 0$ AND $rate_{PT} = 0$	——————————————	climb at X ft/min
msd < 500 feet AND $rate_H = 0$ AND $rate_{PT} > 0$	climb at X ft/min	climb at X ft/min
msd < 500 feet AND $rate_H = 0$ AND $rate_{PT} < 0$	——————————————	climb at X ft/min
msd < 500 feet AND $rate_H > 0$ AND $rate_{PT} = 0$	maintain current heading and rate	climb at X ft/min
msd < 500 feet AND $rate_H < 0$ AND $rate_{PT} = 0$	fly level	climb at X ft/min
msd < 500 feet AND $rate_H > 0$ AND $rate_{PT} > 0$	climb at X ft/min	climb at X ft/min
msd < 500 feet AND $rate_H > 0$ AND $rate_{PT} < 0$	——————————————	climb at X ft/min
msd < 500 feet AND $rate_H < 0$ AND $rate_{PT} > 0$	climb at X ft/min	climb at X ft/min
msd < 500 feet AND $rate_H < 0$ AND $rate_{PT} < 0$	fly level	climb at X ft/min

	$alt_H < alt_{PT}$ AND $alt_{PT} - alt_H \geq 500$ feet	$alt_H < alt_{PT}$ AND $alt_{PT} - alt_H < 500$ feet
msd $\geq$ 500 feet	maintain current heading and rate	maintain current heading and rate
msd < 500 feet AND rate$_H$ = 0 AND rate$_{PT}$ = 0	——————————————————	dive at X ft/min
msd < 500 feet AND rate$_H$ = 0 AND rate$_{PT}$ > 0	——————————————————	dive at X ft/min
msd < 500 feet AND rate$_H$ = 0 AND rate$_{PT}$ < 0	dive at X ft/min	dive at X ft/min
msd < 500 feet AND rate$_H$ > 0 AND rate$_{PT}$ = 0	fly level	dive at X ft/min
msd < 500 feet AND rate$_H$ < 0 AND rate$_{PT}$ = 0	maintain current heading and rate	dive at X ft/min
msd < 500 feet AND rate$_H$ > 0 AND rate$_{PT}$ > 0	fly level	dive at X ft/min
msd < 500 feet AND rate$_H$ > 0 AND rate$_{PT}$ < 0	fly level	dive at X ft/min
msd < 500 feet AND rate$_H$ < 0 AND rate$_{PT}$ > 0	——————————————————	dive at X ft/min
msd < 500 feet AND rate$_H$ < 0 AND rate$_{PT}$ < 0	dive at X ft/min	dive at X ft/min

)

Quantity "X" appearing in the preceding text messages is computed as:

$$X = \frac{(500 - msd)}{t_{msd}}$$

Entries marked with dashed lines denote conditions that are not physically possible.

3.2. ACTIVITIES

3.2.1. Update_ATD

This activity invokes the Display presentation to display changes in potential threat collision attributes.

    demand activity
        name: Update_ATD

context: (potential_threat, true)
starting event: @T(!radar_msg!)
presentation: ATD(potential_threat)

3.2.2. Update_Aircraft_Display_Symbol

This activity invokes the Display presentation to display partition changes in potential threat aircraft.

demand activity
    name: Update_Aircraft_Display_Symbol
    context: (potential_threat, true)
    starting event: @T(<ADS.ID_Shape.Partition > [potential_threat])
                    @F(<ADS.ID_Shape.Partition > [potential_threat])
    presentation: ATD(potential_threat)

< if there exists A ∈ CWS such that A.Response.Alarm then >

3.2.3. Ring_Audible_Alarm

This activity invokes the Audible_Alarm presentation to initiate the audible alarm.

    < forall A in Collision_Warning_Situation >
        < if A.Response.Alarm then >
            demand activity
                name: Ring_Audible_Alarm. < A.CWS_Name >
                context: (potential_threat, true)
                < if A.Partition = ALL then >
                    starting event: @T(< A.CWS_Def > [potential_threat])
                < else >
                    starting event: @T(< A.CWS_Def > [potential_threat])
                                  when partition = < A.Partition >
                < endif >
                presentation: Audible_Alarm(potential_threat)
        < endif >
    < endfor >

The Audible_Alarm presentation is activated only when the potential threat transitions from a lower severity collision warning situation to one of higher severity.

< endif >

< if there exists A ∈ CWS such that A.Response.ATC_Msg then >

3.2.4. Send_ATC_Msg

This activity invokes the ATC_Msg presentation to send a message to the nearest air traffic control center.

    < forall A in Collision_Warning_Situation >
        < if A.Response.ATC_Msg then >
            demand activity

```
                name: Send_ATC_Msg. <A.CWS_Name>
                context: (potential_threat. true)
                <if A.Partition = ALL then>
                    starting event: @T(<A.CWS_Def> [potential_threat])
                <else>
                    starting event: @T(<A.CWS_Def> [potential_threat])
                                            when partition = <A.Partition>
                <endif>
                presentation:  ATC_Msg(potential_threat)
        <endif>
    <endfor>

<endif>
```

< if there exists A ∈ CWS such that A.Response.Inter_Air_Msg then >

3.2.5. Send_Inter_Air_Msg

This activity invokes the Inter_Air_Msg presentation to send a message to the appropriate potential threat.

```
    <foreach A in Collision_Warning_Situation>
        <if A.Response.Inter_Air_Msg then>
            demand activity
                name: Send_Inter_Air_Msg. <A.CWS_Name>
                context: (potential_threat, true)
                <if A.Partition = ALL then>
                    starting event: @T(<A.CWS_Def> [potential_threat])
                <else>
                    starting event: @T(<A.CWS_Def> [potential_threat])
                                            when partition = <A.Partition>
                <endif>
                presentation:  Inter-Air_Msg(potential_threat)
        <endif>
    <endfor>

<endif>
```

3.2.6. Send_Corrective_Message

This activity invokes the Corrective_Action_Msg presentation to display a corrective action advisory message on the host aircraft's ATD.

```
    <forall A in Collision_Warning_Situation>
        <if A.Response.Corrective_Msg then>
            demand activity
                name: Send_Corrective_Msg. <A.CWS_Name>
                context: (potential_threat. true)
                <if A.Partition = ALL then>
```

     starting event: @ T(< **A.CWS_Def** > [potential_threat])
   < **else** >
     starting event: @T(< **A.CWS_Def** > [potential_threat])
          when partition = < **A.Partition** >
   < **endif** >
   presentation: Corrective_Action_Msg
 < **endif** >
< **endfor** >

4. TIMING REQUIREMENTS

Periodic and demand functions are separated because the relevant timing information is different.

4.1. TIMING REQUIREMENTS FOR PERIODIC FUNCTIONS

None

4.2. TIMING REQUIREMENTS FOR DEMAND FUNCTIONS

Function Name	Maximum Delay to Completion
Update_Aircraft_Display_Symbol	250 ms
Update_ATD	250 ms

< **forall A in Collision_Warning_Situation** >
  < **if A.Response.Alarm then** >

Ring_Audible_Alarm. < A.CWS_Name >	250 ms

  < **endif** >
  < **if A.Response.ATC_Msg then** >

Send_ATC_Msg. < A.CWS_Name >	250 ms

  < **endif** >
  < **if A.Response.Inter_Air_Msg then** >

Send_Inter_Air_Msg. < A.CWS_Name >	250 ms

  < **endif** >
  < **if A.Response.Corrective_Msg then** >

Send_Corrective_Msg. < A.CWS_Name >	250 ms

  < **endif** >
< **endfor** >

Local Dictionary

!id_definition!

A predicate that defines the criteria for an identified potential threat. The predicate is written in terms of values of flight characteristics.

!radar_msg!

The event that occurs when the ATD/CWM system receives another Radar_Msg from the radar device.

!transponder_code!

A four-digit integer code in the range 0000 ... 7777 excluding the following reserved codes.

7500
7600-7677
7700-7777

The last two digits should always read 00.

4-22

This page intentionally left blank.

5. ATD/CWM PROCESS REQUIREMENTS

1. PROCESS SPECIFICATION

This section describes the work products, activities, and process of application engineering for the ATD/CWM domain. The activities and process describe the requirements for an application engineering environment that supports the application engineer's decision-making process. The ATD/CWM Application Engineering Process produces a product that is comprised of one or more work products (both deliverable and intermediate).

WORK PRODUCTS

The work products produced by the ATD/CWM application engineering process are:

- Application Model.

- Executable software written in Ada and C.

- DOD-STD-2167A Software Requirements Specification (SRS).

- DOD-STD-2167A Interface Requirements Specification (IRS).

- DOD-STD-2167A Software Design Document (SDD).

APPLICATION ENGINEERING PROCESS AND ACTIVITIES

Figure 5-1 (on the following page) shows the ATD/CWM Application Engineering Process. The dashed boundary delineates the specification portion of this process. The bullet symbols (●) represent choices where the application engineer can perform any of the activities indicated by the arrows.

The following paragraphs describe the activities of the Application Engineering Process. A representation of the forms used by the application engineer follows each activity.

Step 1. Define Application Model Name

The application engineer must first specify a name for his Application Model. The name is entered in the following form under the **Value** column.

Decision	Mnemonic	Value
Application Model Name	Model_Name	alphanumeric (case-sensitive; maximum 64 characters in length)

Specification

Define Application Model Name

Define Host Aircraft
Characteristics

Define Potential Threat
Characteristics

Define Collision
Warning Situation

Define Collision
Warning Situation
Characteristics

Define Situation Response

Define Situation Display

Define Alarm
Characteristics

Validate the Application Model

Assess the Application Model

Generate the Application from
the Application Model

Runtime Validation

Key:

| Activity |

Product Flow

Information

Specification Activity Boundry

Figure 5-1. Air Traffic Display/Collision Warning Monitor Application Engineering Process

The name must uniquely identify this Application Model.

Having entered the Application Model name, the application engineer performs any one of the following steps. However, he must perform all of them before he has completed the Application Model.

- Define Host Aircraft Characteristics

- Define Potential Threat Characteristics

- Define Collision Warning Situation

Step 2. Define Host Aircraft Characteristics

The application engineer specifies the surveillance area radius, the icon shape for the host aircraft, and the ATC_Msg message format. These values are entered in the following form.

Decision	Mnemonic	Value
Host_Aircraft Characteristics	—————————————	—————————————
Surveillance Area	Surveillance_Area	numeric (range 10 to 300)
Host Aircraft Shape	Host_Aircraft_Shape	enumeration (**circle, square, triangle**)
ATC_Message_Mode	Message_Mode	enumeration (A. C)

Step 3. Define Potential Threat Characteristics

The application engineer specifies characteristics unique to potential threats. These characteristics are the criteria for distinguishing between **identified** and **unidentified** aircraft and the icon shape for **identified** and **unidentified** aircraft.

Decision	Mnemonic	Value
Potential Threat Characteristics	—————————————	—————————————
Identification Requirements	ID_Req	set (**airspeed, altitude**)
Shape of Identified Aircraft	ID_Shape	enumeration (**circle, square, triangle**)
Shape of Unidentified Aircraft	UID_Shape	enumeration (**circle, square, triangle**)

A potential threat is considered **identified** when all of its attributes selected in the Identification Requirements field are known.

Step 4. Define Collision Warning Situation

The application engineer specifies the name of a collision warning situation. He then performs four more individual steps, in any order, to specify the characteristics of this situation, the response that the ATD/CWM system performs when this situation is recognized, audible alarm characteristics (if necessary), and how this situation is displayed to the pilot in the host aircraft on the ATD. These four steps are listed below.

1. Define Collision Warning Situation Definition. The application engineer performs this step to specify the characteristics (e.g., distance, severity) of this collision warning situation.

2. Define Situation Response. The application engineer performs this step to specify the actions the ATD/CWM system performs when it recognizes this collision warning situation.

3. Define Situation Display. The application engineer performs this step to specify the icon color, fill, and blink characteristics of the host aircraft, identified potential threats, and unidentified potential threats.

4. Define Alarm Characteristics. The application engineer performs this step to specify the audible alarm characteristics. This step is not performed if the application engineer does not select the Alarm response for this collision warning situation.

Decision	Mnemonic	Value
Collision Warning Situation	——————	————————
Collision Warning Situation Name	CWS_Name	alphanumeric (case-insensitive with a maximum length of 64 characters)
Situation Definition	——————	————————
Situation Aircraft Partition	Partition	enumeration (ID, UID. ALL)
Situation Severity	Severity	numeric (range 0.00 to 1.00. resolution 0.01)
Situation Flight Characteristics	——————	————————
Time	——————	————————
Min	Time_Min	numeric (range 1.0 to 300.0. resolution 0.1)
Max	Time_Max	numeric (range 1.0 to 300.0. resolution 0.1)
Range	——————	————————
Min	Range_Min	numeric (range 0.0 to X where X is the value chosen for the surveillance area)
Max	Range_Max	numeric (range 0.0 to X where X is the value chosen for the surveillance area)
Situation Response	——————	————————
Response to ATC	ATC_Msg	enumeration of (True, False)
Response to other Aircraft	Inter_Air_Msg	enumeration of (True, False)
Corrective Action Response	Corrective_Msg	enumeration of (True, False)
Alarm	Alarm	enumeration of (True, False)
Code	Code	numeric (exactly 4-digit integer; range 0000–7777 excluding codes 7500 7600 through 7677 7700 through 7777 Last two digits must be 00.
Alarm Characteristics	——————	————————

Decision	Mnemonic	Value
Pitch	Alarm_Pitch	numeric (range 1000 through 10,000)
Duration	Alarm_Duration	numeric (range 0.01 to 10.00; resolution 0.01)
Situation Display	————	————————————
Color of Host Aircraft	Host_Color	enumeration (**red, yellow, pink, orange, blue, green, white, black, purple, indigo, violet**)
Color of Identified Potential Threats	ID_Color	enumeration (**red, yellow, pink, orange, blue, green, white, black, purple, indigo, violet**)
Blinking Identified Potential Threats	ID_Blink	enumeration (**True, False**)
Fill Identified Potential Threats	ID_Fill	enumeration (**True, False**)
Color of Unidentified Potential Threats	UID_Color	enumeration (**red, yellow, pink, orange, blue, green, white, black, purple, indigo, violet**)
Blinking Unidentified Potential Threats	UID_Blink	enumeration (**True, False**)
Fill Unidentified Potential Threats	UID_Fill	enumeration (**True, False**)

If the named collision warning situation does not exist, it is created. The collision warning situation name NORMAL is reserved and cannot be specified by the application engineer, including all upper- and lower-case variations.

If the named collision warning situation exists, the reminder of this form will contain previously entered values.

If the application engineer choses **true** for Alarm, he must also specify the alarm characteristics pitch and duration. Otherwise, he can ignore these characteristics.

If the application engineer choses **true** for **Response to ATC**, he must chose a value for the **Code** decision.

Step 4 is repeated as often as there are collision warning situations to specify. A separate form is completed for each collision warning situation.

Step 5. Validate the Application Model

The application engineer uses this activity to validate the Application Model. This step can only be performed after the application engineer completes his Application Model. Furthermore, he must validate it before he can use it to generate the application (see *Generate the Application from the Application Model*). The following checks are applied during validation:

- The application engineer has defined at least one collision warning situation.

- The application engineer has defined values for all fields for every collision warning situation.

- The application engineer has marked the Corrective_Msg decision as **true** for at least one collision warning situation.

- The application engineer has specified a value for the surveillance area range.

- The application engineer has specified a value for the ATC_Message format if, and only if, there is at least one collision warning situation response which has the **Response to ATC** decision marked **true**.

- The value space constraints for time and range are enforced when used in the collision warning situation characteristics. For each collision warning situation in which Time is specified, Time_Min $\leq$ Time_Max. For each collision warning situation in which Range is specified, the following checks are done:

    - Range_Min $\leq$ Range_Max

    - Range_Min $\leq$ Surveillance Area

    - Range_Max $\leq$ Surveillance Area

- The application engineer has specified mutually exclusive icon shapes for the host aircraft, identified potential threats, and unidentified potential threats.

- The collision warning situations cover the entire surveillance area and none of them overlap.

Step 6. Assess the Application Model

The application engineer uses this activity to assess the Application Model. The following check is performed during this activity:

- How quickly would the ATD/CWM system respond to a detected collision warning situation?

Step 7. Generate the Application from the Application Model

The application engineer uses this activity to generate an application from a validated Application Model.

Step 8. Runtime Validation

The application engineer uses this activity to validate other characteristics of his ATD/CWM system by performing the following checks while his ATD/CWM system is executing:

- The ATD/CWM system performs the desired actions in response to detected collision warning situations.

- Each aircraft in the surveillance area is displayed with the desired identifying icon.

- The ATD/CWM system recognizes each collision warning situation.

2. APPLICATION MODELING NOTATION SPECIFICATION

FORM

The form presentation paradigm is a region consisting of a set of label/field pairs. The labels are text describing the field content and nature. The individual fields are typed. They specify constraints that

exist for the given field. The application engineer is notified if any of the individual field constraints are violated. He is allowed to create, modify, or delete information associated with any one of the individual fields. There is also a paradigm for moving between the individual fields of the form.

Decision	Mnemonic	Value

The form name identifies a particular application engineering form. The paradigm for forms is fixed as Form. The form is made up of one or more fields which have the following parameters: Decision, Mnemonic, and Value. The parameters of this form specification table are:

- *Decision.* The Decision identifies a particular field in the form. This is a text string that will be used to label the associated field.

- *Mnemonics.* The Mnemonic identifies a shorthand abbreviation of the Decision.

- *Value.* Each Value has a specific type (described in the form itself) which is one of the following:

    - *Alphanumeric.* An alphanumeric field allows the application engineer to specify text consisting of alphabetic and numeric characters. An alphanumeric field must begin with an alphabetic character. Constraints on the number of allowed characters are defined in the Decision Model. Additional constraints such as case-sensitivity and variable-length may also be included for this field type.

    - *Enumeration.* An enumeration field consists of a discrete set of choices. The application engineer can select individual choices from the legal list of choices (i.e., **true** and **false** as examples of a boolean decision).

    - *Numeric.* A numeric field allows the application engineer to specify a numeric value. Value constraints for numeric fields are obtained from the Decision Model. Typical constraints include restricted ranges of numbers, integer-only, and real numbers with an established number of significant digits.

    - *Set.* An enumeration field consists of a discrete set of choices that are presented to the application engineer. He can select multiple choices from the legal list of choices.

    - *Text.* A text field allows the application engineer to specify arbitrary free-form text (i.e., any printable character). Additional constraints such as case-sensitivity may also be included for this field type. There are no predefined maximum number of characters specified for a text field.

- Each field has an associated decision from the Decision Model. When the application engineer enters information in a field, it is associated with the corresponding Decision Model decision.

This page intentionally left blank.

6. ATD/CWM PRODUCT DESIGN

NOTE: The ADARTS (Software Productivity Consortium 1991b) method, as modified in Section A.6 to fit the requirements method, was used to develop the ATD/CWM Product Design.

Product Design consists of three subproducts: Product Architecture, Component Design, and Generation Design.

1. PRODUCT ARCHITECTURE

The Product Architecture for the ATD/CWM domain is described by three structures: an adaptable information hiding structure, an adaptable process structure, and an adaptable dependency structure. The description of each structure consists of two parts: an interface and a textual/graphical form of the structure. The interface consists of three parts: instantiation parameters, instantiation constraints, and a local dictionary The instantiation parameters describe what adaptations are possible for the given structure. The value space and a definition are provided for each instantiation parameter. The instantiation constraints describe any relations that exist among the instantiation parameters that must be satisfied to obtain a valid structure for a particular family member. The local dictionary defines terms used in the instantiation parameters or constraints.

ADAPTABLE INFORMATION HIDING STRUCTURE

Instantiation Parameters

Parameter Name	Value Space	Definition
alarm	boolean	A **true** value means that the components to support the Audible_Alarm must be included in the information hiding structure. Otherwise, this parameter's value is **false**.
atc_msg	boolean	A **true** value means that the components supporting transmission of an ATC_Msg to air traffic control must be included in the information hiding structure. Otherwise, this parameter's value is **false**.
inter_air_msg	boolean	A **true** value means that the components supporting transmission of an Inter_Air_Msg to a potential threat must be included in the information hiding structure. Otherwise, this parameter's value is **false**.
temp_buffer	list of !buffer!	Each record in this list contains the name, mnemonic, and description of an instance of the Temporary_Data_Buffers module.

Instantiation Constraints – none

Local Dictionary

!buffer!   record of   (
                                                name : identifier,
                                                mnemonic : identifier,
                                                description : text
)

The graphical form of the adaptable information hiding structure is represented in Figures 6-1 through 6-4. Descriptions of the modules follow the figures.
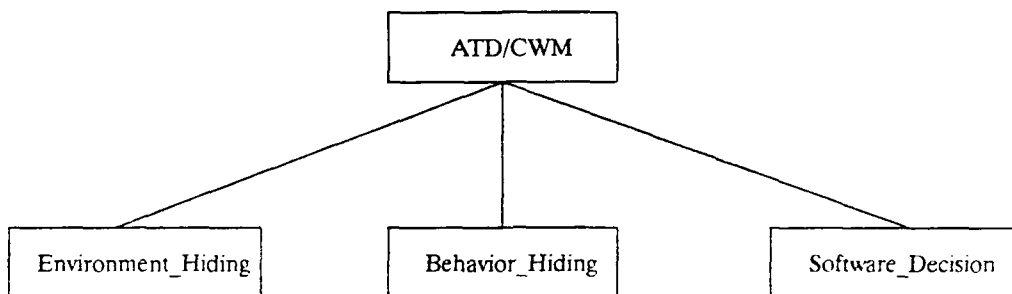
Figure 6-1. Top-Level of the Air Traffic Display/Collision Warning Monitor Information Hiding Structure
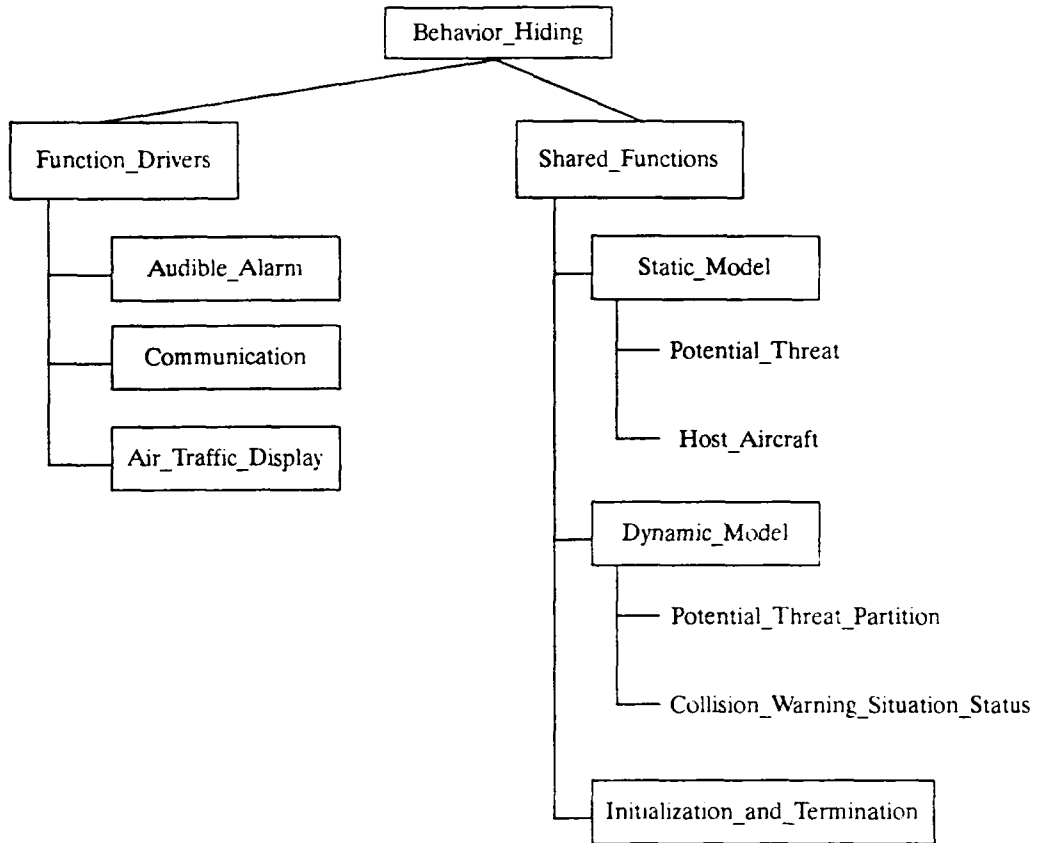
Figure 6-2. Decomposition of the Behavior_Hiding Module
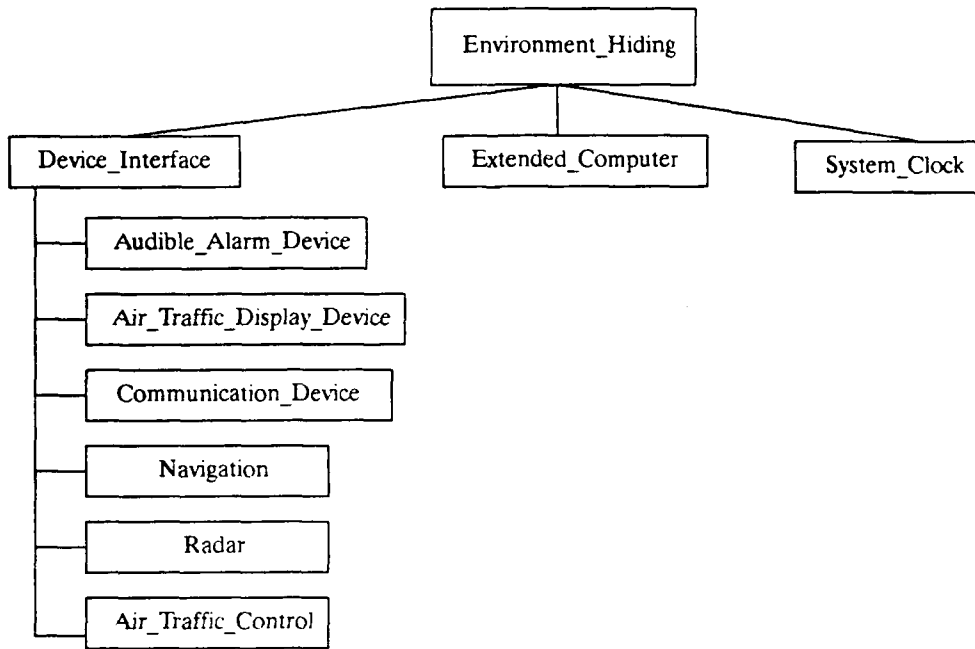
Figure 6-3. Decomposition of the Environment_Hiding Module
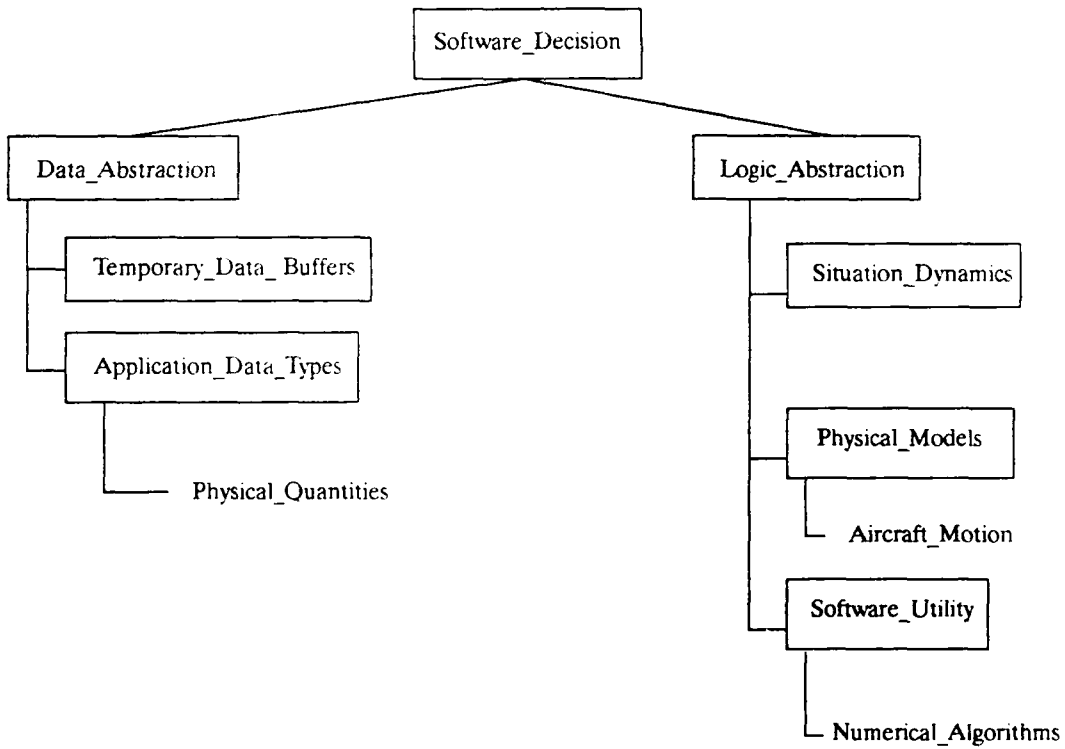
Figure 6-4. Decomposition of the Software_Decision Module

Textual description of the adaptable information hiding structure.

1. Environment_Hiding (EH)

The Environment_Hiding module includes the programs that need to be changed if any part of the hardware and software operating environment of the ATD/CWM system changes or is replaced by another part that presents a different interface but has the same general capabilities. This module implements a virtual environment that the rest of the ATD/CWM system uses. The primary hidden decisions of this module are the interfaces provided by the actual devices and software systems in the ATD/CWM environment. The secondary hidden decisions are the algorithms and data structures used to implement the virtual environment.

1.1. Extended_Computer (EC)

The Extended_Computer module hides the characteristics of the processing environment that are considered likely to change if the product set is moved to another computer, operating system. or a different language or language compiler. This module provides an integrated abstraction of processor. operating system, and language capabilities.

1.2. Device_Interface (DI)

The Device_Interface module contains the programs that need to change if one or more of the devices with which the ATD/CWM software must interact are replaced with a device having a different hardware/software interface but the same general capabilities.

1.2.1. System_Clock (CLK)

The System_Clock module encapsulates how software determines the current time and date. The primary hidden decision is the hardware/software interface to the hardware clock.

< if alarm then >

1.2.2. Audible_Alarm_Device (AAD)

The Audible_Alarm_Device module encapsulates the hardware/software interface to the audible alarm. Its primary hidden decisions are the value encoding of the frequency and duration to the device.

< endif >

1.2.3. Air_Traffic_Display_Device (ATDD)

The Air_Traffic_Display_Device module encapsulates the hardware/software interface to the display. Its primary hidden decisions are the particular sequence of operations necessary to enable and position various icon symbols; the methods for manipulating icon color, shape, shade, and blink characteristics; the method for removing an icon from the display; and the method for writing text to the display.

< if atc_msg OR inter_air_msg then >

1.2.4. Communication_Device (CD)

The Communication_Device module encapsulates the hardware/software interface to the communication device. Its primary hidden decision is how to transmit a string of characters to either the nearest air traffic control center or specified potential threat.

< endif >

1.2.5. Navigation (NAV)

The Navigation module encapsulates the hardware/software interface to the host aircraft navigation device. The primary hidden decisions are how to obtain host aircraft raw data for altitude, airspeed, ground_track, latitude, and longitude; the scale and format of these input data items; and the device-dependent operations that must be applied to convert the raw data to the internal format of the ATD/CWM system.

1.2.6. Radar (RADAR)

The Radar module encapsulates the hardware/software interface to the radar. The primary hidden decisions are how to obtain raw data for the aircraft_identification, sweep, relative bearing, range, and timestamp; the scale and format of these input data items; and the device-dependent operations that must be applied to convert the raw data to the internal format of the ATD/CWM system.

1.2.7. Air_Traffic_Control (ATC)

The Air_Traffic_Control module encapsulates the hardware/software interface to the ATC device. Its primary hidden decisions are how to obtain raw data for the aircraft_identification, altitude, airspeed, ground_track, and range; the scale and format of these input data items; and the device-dependent operations that must be applied to convert the raw data to the internal format of the ATD/CWM system.

2. Behavior_Hiding (BH)

The Behavior_Hiding module contains all the software that would need to be changed if the externally visible, required behavior of the system were to change without an attending change in the hardware. The primary hidden decision of the module is the rules for producing the required system outputs. The secondary hidden decision is the algorithms and internal data structures used to implement the required behavior.

2.1. Function_Drivers (FD)

The Function_Drivers module consists of a set of modules called function drivers. Each function driver is the sole controller of a set of closely related outputs. The primary hidden decisions of this module are the rules determining the values of these outputs and the rules determining when these values are computed.

< if alarm then >

2.1.1. Audible_Alarm (AA)

The hidden decisions of the Audible_Alarm module are to determine the frequency and duration at which to initiate the audible alarm for a specific collision warning situation.

< endif >

< if atc_msg OR inter_air_msg then >

2.1.2. Communication (COMM)

The hidden decision of the Communication module is how the content of the Communication messages are determined for a specific collision warning situation.

< endif >

2.1.3. Air_Traffic_Display (ATD)

The hidden decisions of the Air_Traffic_Display module are when to display aircraft status, where to place aircraft symbols, what information to display, and the content of the corrective action message.

2.2. Shared_Functions (SF)

Because some behavior is common to several behavior modeling modules, it is expected that if there is a change in that aspect of the behavior, it will affect all of the functions that share it. Consequently, a set of modules have been identified each of which hides an aspect of the behavior that may be shared by two or more other behavior hiding modules.

2.2.1. Static_Model (SM)

The hidden decision of the Static_Model module is how to represent and manipulate the static model of ATD/CWM systems.

2.2.1.1. Potential_Threat (PT)

The Potential_Threat module models potential threats in an ATD/CWM system. Potential threats have properties of altitude, aircraft_identification, airspeed, ground_track, range, relative_bearing, rate, and cws_status. The hidden decisions of this module are the internal representation of the properties, algorithms for manipulating them, and how to determine the values for potential threat properties.

2.2.1.2. Host_Aircraft (HA)

The Host_Aircraft module models the host aircraft in an ATD/CWM system. The host aircraft has properties of altitude, aircraft_identification, airspeed, location, ground track, rate, and cws_status. The hidden decisions of this module are the internal representation of these properties, algorithms for manipulating them, and how to determine the values for these properties.

2.2.2. Dynamic_Model (DM)

The Dynamic_Model module hides a model of how a collision warning situation changes as aircraft move on predicted flight paths.

2.2.2.1. Collision_Warning_Situation_Status (CWSS)

The hidden decisions of the Collision_Warning_Situation_Status module are how to determine the collision warning situation status of potential threats and the host aircraft.

2.2.2.2. Potential_Threat_Partition (PTP)

The hidden decision of the Potential_Threat_Partition module is how to determine the potential threat partition.

2.2.3. Initialization_and_Termination (IT)

The hidden decision of the Initialization_and_Termination module is how system operation is initiated and terminated.

3. Software_Decision (SD)

The Software_Decision module hides software design decisions that are based on mathematical theorems, physical facts, and programming considerations such as algorithmic efficiency and accuracy. The hidden decisions of this module are not described in the product specification. This module differs from the other modules in that both the hidden decisions and the interfaces are determined by software designers. Changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

3.1. Data_Abstraction (DA)

The Data_Abstraction module provides data types, including persistent data object collections, that are useful in the ATD/CWM domain. The primary hidden decisions of the module are the representation of the data and the representation of the algorithms used to implement the data types. Units of measure are part of the representation and are hidden. Where necessary, the modules provide conversion operations which deliver or accept values in specified units.

3.1.1. Temporary_Data_Buffers (TDB)

The Temporary_Data_Buffers module encapsulates details about buffers used to communicate information between programs. The primary hidden decisions are the size of the buffers, are they fixed or vary in size, are they stored contiguously in memory or not, what to do when a buffer is full or empty, are they loosely- or tightly-coupled, and are the regimes for temporary storage first-in/first-out, last-in/first-out.

```
< forall B in temp_buffer >
    < B.name > ( < B.mnemonic > )
    < B.description >
< endfor >
```

3.1.2. Application_Data_Types (ADT)

The Application_Data_Types module provides abstract data types useful for the ATD/CWM domain. The primary hidden decisions are the representation of the data type and the representation of the algorithms used to manipulate them. Where necessary, conversion operations are also provided.

3.1.2.1. Physical_Quantities (PQ)

The Physical_Quantities module encapsulates details about data types used to represent physical quantities such as distance and velocity. The hidden decisions of this module are the representation of these data types; the use of range and resolution to determine representation; algorithms for performing operations; and conversions required when two quantities of the same data type are not represented in the same way.

3.2. Logic_Abstraction (LA)

The Logic_Abstraction module implements models that derive values based on relationships among other values. The primary hidden decision of this module are the algorithms used to derive the values.

3.2.1. Situation_Dynamics (SD)

The hidden decisions of the Situation_Dynamics module are how physical models can be put together to predict a future situation starting from a known state history.

3.2.2. Physical_Models (PM)

The software requires estimates of quantities that cannot be measured directly but can be computed from observables using mathematical models. The primary hidden decisions of the Physical_Models module are the models and the implementation of those models.

3.2.2.1. Aircraft_Motion (AM)

The Aircraft_Motion module encapsulates details of the model of an aircraft's motion which are used to calculate aircraft position and altitude from observable inputs. The primary hidden decision of this module is the equation of motion.

3.2.3. Software_Utility (SU)

The Software_Utility module contains those utility routines that would otherwise have to be written by more than one other module. The hidden decisions of this module are the data structures and algorithms used to implement the programs.

3.2.3.1. Numerical_Algorithms (NA)

The Numerical_Algorithms module provides mathematical service routines needed by more than one module within the system. These functions include services for data manipulations such as square root and trigonometric functions. The hidden decisions of this module are the algorithms implementing the functions.

ADAPTABLE PROCESS STRUCTURE

Instantiation Parameters

Parameter Name	Value	Definition
cws	list of !cws_info!	Each record defines the set of responses performed by the ATD/CWM system for the specified collision warning situation.

Instantiation Constraints – none

Local Dictionary

!cws_info!   record of (
                                  cws_name : identifier,
                                  alarm : boolean,
                                  atc_msg : boolean,
                                  inter_air_msg : boolean,
                                  corrective_msg : boolean
)

The adaptable process structure is described in two parts. First, the atomic entities used to derive the adaptable process structure are listed. Second, the adaptable process structure is described in terms of atomic entity groupings and rationale.

- Starting Point

The initial set of atomic entities (i.e., they cannot be subdivided) were identified from the Product Requirements using the heuristics discussed below.

One atomic entity for each entity of each primary static model class.

Atomic Entity	How Many
Host_Aircraft	1
Potential_Threat	1

One atomic entity for each device.

Atomic Entity	How Many
Navigation	1
Radar	1
Air_Traffic_Control	1
Audible_Alarm_Device	1
Air_Traffic_Display_Device	1
Communication_Device	1

One atomic entity for each device input mapping.

Atomic Entity	How Many
Process_Navigation	1
Process_Radar	1
Process_ATC	1

One atomic entity for each activity.

Atomic Entity	How Many
Update_ATD	1
Update_Aircraft_Display_Symbol	1
Ring_Audible_Alarm	0 or more
Send_ATC_Msg	0 or more
Send_Inter_Air_Msg	0 or more
Send_Corrective_Msg	1 or more

One atomic entity for each Dynamic Model process.

Atomic Entity	How Many
Time_To_Impact	1
Minimal_Separation_Distance	1
Potential_Threat_CWS	1
Potential_Threat_Partition	1
Host_Aircraft_CWS	1
Update_Relative_Bearing	1

- Task Structuring

Update_Host_Aircraft_Information

Composed Of	Criteria
Navigation Host_Aircraft Process_Navigation	Synchronous I/O device (The navigation device periodically updates the data and transmits it to the ATD/CWM system). Periodic event (need for up-to-date host aircraft location within the resolution specified by the aircraft position algorithm).

Update_Potential_Threat_Information

Composed Of	Criteria
Potential_Threat Potential_Threat_CWS Update_Relative_Bearing Potential_Threat_Partition Time_To_Impact Minimal_Separation_Distance	Entity modeling (for each potential threat, updating the information must be accomplished). Sequential cohesion (when new information is received for each potential threat, its collision warning situation status must be recomputed).

Get_Radar_Information

Composed Of	Criteria
Radar Process_Radar	Sequential cohesion.

Get_ATC_Information

Composed Of	Criteria
Air_Traffic_Control Process_ATC	Sequential cohesion.

Update_ATD

Composed Of	Criteria
Air_Traffic_Display_Device	Passive I/O device.

Update_Aircraft_Display_Symbol

Composed Of	Criteria
Update_Aircraft_Display_Symbol	To be determined

< if there exists C ∈ cws such that C.alarm then >

Output_Alarm

Composed Of	Criteria
Audible_Alarm_Device	Asynchronous I/O device. Buffer the ATD/CWM system from the Audible Alarm device driver.

< endif >

< if there exists C ∈ cws such that (C.atc_msg OR C.inter_air_msg) then >

Output_Communication

Composed Of	Criteria
Communication_Device	Asynchronous I/O device. Buffer the ATD/CWM system from the Communication device driver.

< endif >

< forall C in cws >

Collision_Warning_Situation_ < C.cws_name >

Composed Of	Criteria
< if C.alarm then > Ring_Audible_Alarm **< endif >** **< if C.atc_msg then >** Send_ATC_Msg **< endif >** **< if C.inter_air_msg then >** Send_Inter_Air_Msg **< endif >** **< if C.corrective_msg then >** Send_Corrective_Msg **< endif >** Update_CWS	Sequential cohesion. Functional cohesion (all responses are involved in the response due to a transition into this collision warning situation). Separation from other tasks which perform the same functionality because of prioritization.

< endfor >

Figure 6-5 shows a graphical representation of the adaptable process structure for the ATD/CWM domain.

KEY:

Loosely coupled message communication
buffer or group of x-related loosely
coupled message communication buffers

(x)

Process or group of x-related processes

Datastore

Event
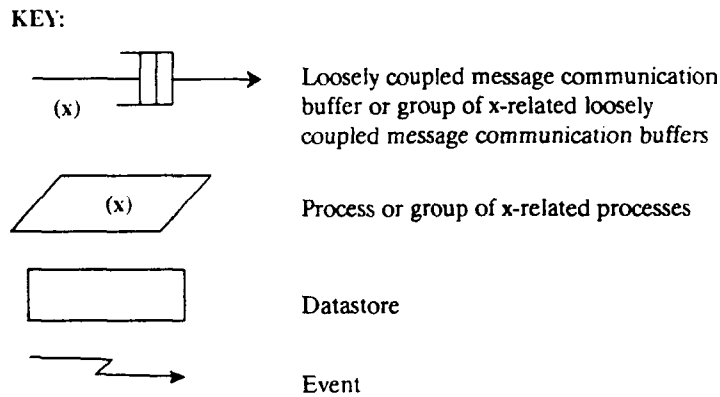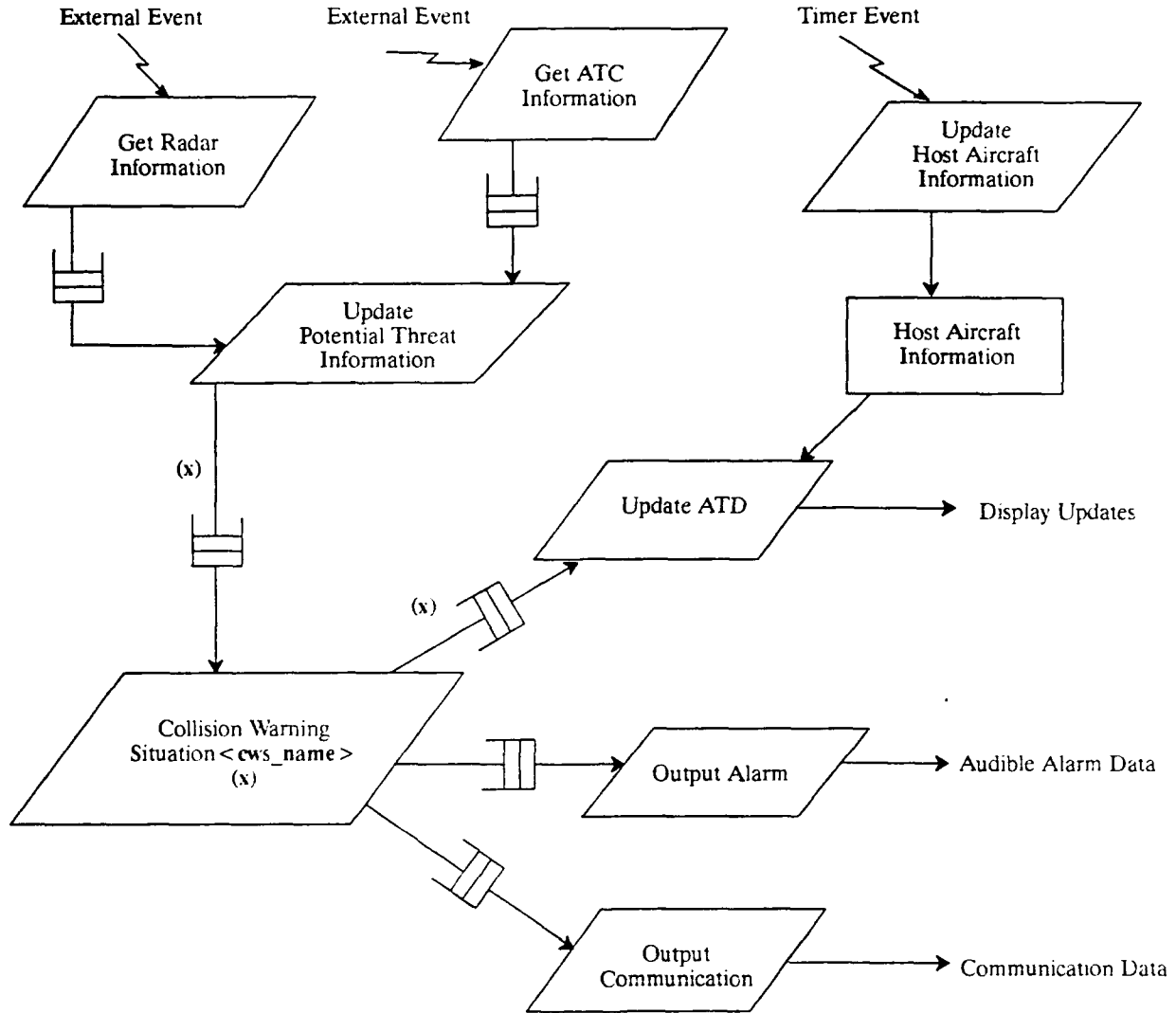
Figure 6-5. Adaptable Task Architecture Diagram for the Air Traffic Display/Collision Warning Monitor Domain

- Code Component – Task Integration

The following table shows which adaptable code components in the adaptable information hiding structure contain the tasks.

Adaptable Code Component	Task
Air_Traffic_Display	Update_ATD
Host_Aircraft	Update_Host_Aircraft_Information
Potential_Threat	Get_Radar_Information Get_ATC_Information < forall C in cws > Collision_Warning_Situation. < C.cws_name > < endfor > Update_Potential_Threat_Information

< if there exists C ∈ cws such that C.alarm then >

Audible_Alarm_Device	Output_Alarm

< endif >

< if there exists C ∈ cws such that (C.atc_msg OR C.inter_air_msg) then >

Communication_Device	Output_Communication

< endif >

Adaptable Dependency Structure

Figure 6-6 depicts the dependency structure for the ATD/CWM domain. The dependency assumptions and Interface Requirements for all leaf adaptable code components are captured in the adaptable code component interface specifications.
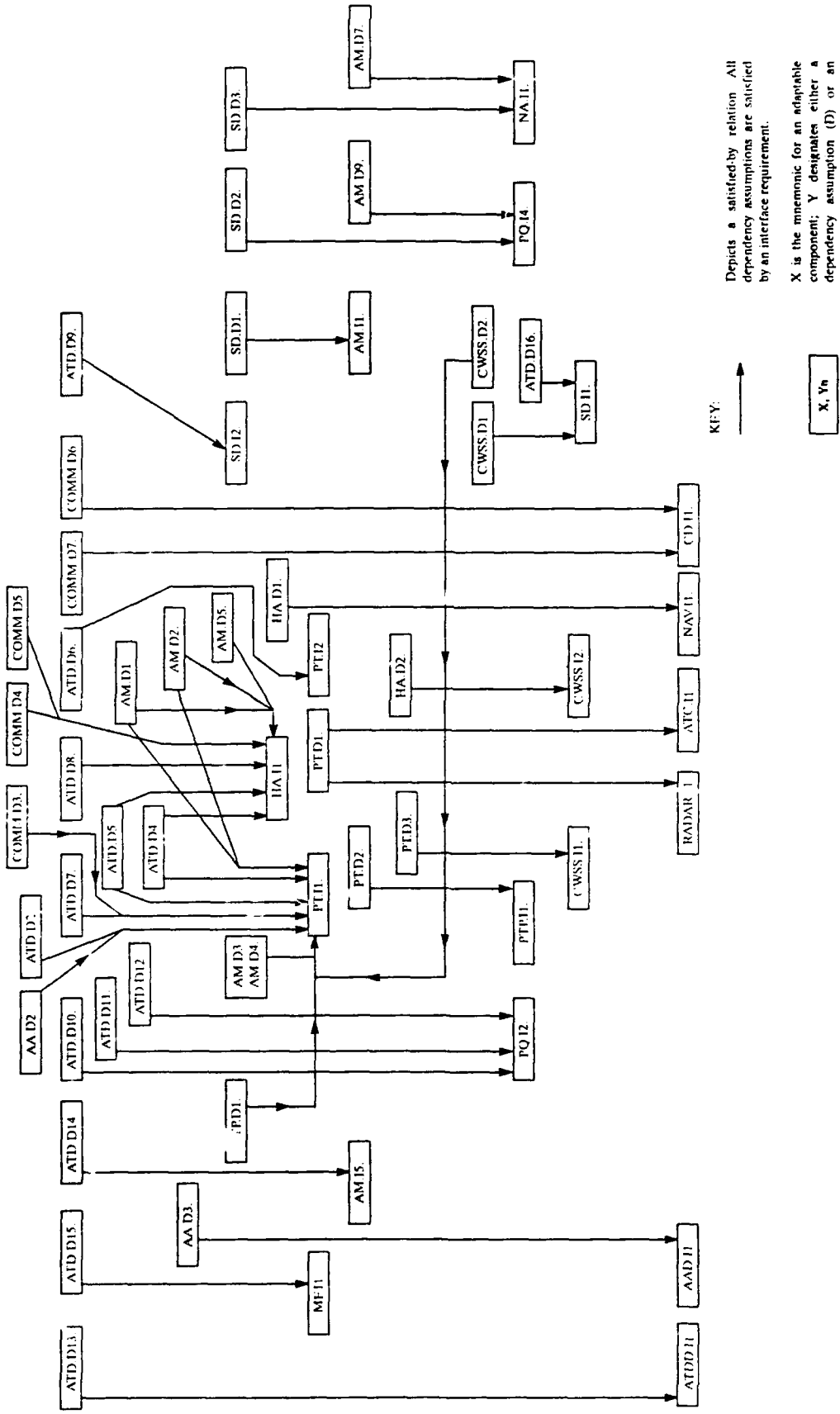
Figure 6-6. Adaptable Dependency Structure

2. COMPONENT DESIGN

Adaptable components are described by an interface which may consists of the following parts:

- *Instantiation Parameters.* Describe what adaptations are possible for the given adaptable component. A parameter name, type, and description are provided for each instantiation parameter.

- *Instantiation Constraints.* Describe any relationships that must be satisfied to obtain a valid component.

- *Assumptions.* Contain a concise statement of unchangeable aspects of the component. The assumptions describe what needs to be true for any implementation to work. The assumptions describe both dependency and interface assumptions.

- *Concrete Operations.* Describes programs that can be called by programs in other components. For each operation, the parameters are described (e.g., name, type, whether it is read only [I], update only [O], or both read and update [IO] and undesired events are listed).

- *Effects.* Describe the externally visible effects of each concrete operation.

- *Events Signalled.* Describe the signals from this component to users. Indicates the occurrence of some state change within this component.

- *Types.* Contain types defined by this component that can be referenced by other components.

- *Local Dictionary.* Defines any terms used in the overall interface description.

- *Undesired Event Dictionary.* Contains definitions of the undesired events that are referred to in the concrete operations.

The interface description for each adaptable code component contains all of these parts. However, the interface description for adaptable documentation components contains only the Instantiation Parameters and Local Dictionary. Adaptable verification and validation components contain only the Instantiation Parameters, Instantiation Constraints, and Local Dictionary.

ADAPTABLE CODE COMPONENTS

1. Aircraft_Motion (AM)

This module contains programs that model aircraft motion. Aircraft location, velocity, and altitude with respect to the earth and airmass are derived from measures of aircraft motion from devices and other physical models modules.

Instantiation Parameters

Parameter	Type	Description
msd	positive	Minimal separation distance as dictated by the FAA such that if two aircraft pass each other within this limit, then a collision has occurred

Instantiation Constraints – none

Assumptions

Dependency Assumption

D1. There is a way to obtain the aircraft's altitude.

D2. There is a way to obtain the aircraft's velocity.

D3. There is a way to obtain the range for a potential threat.

D4. There is a way to obtain the relative bearing for a potential threat.

D5. There is a way to obtain the ground track of the host aircraft.

D6. There is a way to perform arithmetic operations on velocity, altitude, and rate.

D7. There is a way to compute the square root of a numeric quantity.

D8. There is a way to determine how much time has elapsed between successive readings for an aircraft's altitude.

D9. There is a way to compute trigonometric functions.

Interface Requirement

I1. This module must determine the current velocity of an aircraft in the vertical axis.

I2. This module must determine the current velocity of an aircraft's velocity in the X–Y plane.

I3. This module must predict the relative bearing of an aircraft with respect to another one.

I4. This module must allow users to determine the FAA allowed minimal separation distance.

I5. This module must determine the range component that lies in the X–Y plane.

I6. This module must determine the climb_rate of an aircraft.

Concrete Operations

Operation	Parameter	Description	Undesired Events
get_msd	distance:Physical_Quantities.feet:O	!msd!	None
get_range_xy	distance: Physical_Quantities.nautical_mile:I altitude_A:Physical_Quantities.feet:I altitude_B:Physical_Quantities.feet;I range: Physical_Quantities.nautical_mile:O	!range_AB! !alt_A! !alt_B!	None
get_climb_rate	altitude_Y:Physical_Quantities.feet:I time_Y:Physical_Quantities.seconds:I altitude_X:Physical_Quantities.feet:I time_X:Physical_Quantities.seconds:I climb_rate:Physical_Quantities.fpm:O	!alt_Y! !time_Y! !alt_X! !time_X! !rate!	climb_rate_is_infinite
get_velocity_xy	velocity:Physical_Quantities.knots;I rate:Physical_Quantities.fpm: xyvel:Physical_Quantities.knots:O	!velocity! !rate!	None

Effects

get_msd   Returns the FAA minimal separation distance.

get_range_xy   Returns the component of range in the X–Y plane via the following equation.

$$range = (distance^2 - (altitude\_A - altitude\_B)^2)^{0.5}$$

get_climb_rate   Returns the velocity of the specified aircraft in the vertical axis (i.e., climb_rate). The climb_rate is computed as:

$$(altitude\_Y - altitude\_X) / (time\_Y - time\_X)$$

where $time\_Y > time\_X$.

get_velocity_xy   Returns the current velocity of the specified aircraft in the X–Y plane via the following equation.

$$xyvel = (velocity^2 - rate^2)^{0.5}$$

Events Signalled – none

Types – none

Local Dictionary

!alt_A!, !alt_B!   Most recently measured altitude readings for aircraft A and aircraft B, respectively.

!alt_X!, !alt_Y!   Altitude readings for the aircraft at times !time_X! and !time_Y!, respectively.

!msd!   Minimal separation distance dictated by the FAA (i.e., aircraft whose flight paths that intersect each other within this distance are considered to have collided).

!range_AB!   Distance between the aircraft A and B.

!rate!   Vertical rate of change (i.e., climb rate).

!time_X!, !time_Y!   Times at which !alt_X! and !alt_Y! were measured, respectively.

!velocity!   Most recently measured aircraft velocity.

Undesired Event Dictionary

climb_rate_is_infinite

This undesired event occurs when time_Y equals time_X in the "get_climb_rate" operation.

2. Air_Traffic_Control (ATC)

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Dependency Assumption

None

Interface Requirement

I1. This module must provide altitude, aircraft_identification, velocity, ground_track, range, and relative_bearing for a potential threat. It must also provide the timestamp defining when this data was received.

Concrete Operations

Operation	Parameter	Description	Undesired Events

get_atc_message	aircraft_id:string(8):O
	altitude:Physical_Quantities.feet:O
	airspeed:Physical_Quantities.knots:O
	ground_track:
	    Physical_Quantities.degrees:O
	range:
	    Physical_Quantities.nautical_mile:O
	relative_bearing:
	    Physical_Quantities.degrees:O
	timestamp:Physical_Quantities.seconds:O
			None

Effects

Events Signalled – none

Types – none

Local Dictionary

Undesired Event Dictionary – none

3. Air_Traffic_Display (ATD)

Instantiation Parameters

Parameter	Type	Description
icon_shape	!icon_shape!	A record that defines the icon shape to use for potential threats and the host aircraft
display	list of !pt_display!	Each record defines the shape, color. blinking, and fill characteristics for the icon representing a potential threat in the specified collision warning situation.
color	list of !host_color!	A list of records each defining the color of the host aircraft icon for different collision warning situations.
area	identifier	Diameter of the surveillance area.

Instantiation Constraints

1.   Adaptable component Air_Traffic_Display_Device must be instantiated as well.

Assumptions

Dependency Assumption

D1.   There is a way to determine which aircraft needs to be updated on the display.

D2.   There is a way to determine what collision warning situation an aircraft is in relative to the host aircraft.

D3.   There is a way to determine which aircraft has triggered the event requiring the need to have a Corrective_Action_Msg generated.

D4.   There is a way to determine the identification number for a potential threat.

D5.   There is a way to determine the altitude of the host aircraft and potential threat.

D6.   There is a way to determine the rate of the host aircraft and potential threat.

D7.   There is a way to determine the potential threat's partition.

D8.   There is a way to determine the ground track of the potential threat.

D9.   There is a way to determine the ground track of the host aircraft.

D10.   There is a way to determine the distance between the host aircraft and any potential threat.

D11.   There is a way to determine the minimal separation distance between a potential threat and the host aircraft.

D12.   There is a way to compare aircraft rates.

D13. There is a way to compare aircraft ground tracks.

D14. There is a way to compare aircraft altitudes.

D15. There is a way to display aircraft status on the ATD.

D16. There is a way to determine the FAA-allowed minimal separation distance.

D17. There is a way to convert an integer number to a character string.

D18. There is a way to determine how much time elapses (time to intersect) before two aircraft reach a separation minima.

D19. There is a way to initialize the display.

Interface Requirements

I1. This module must allow users to initialize the display.

I2. This module must allow users to format and transmit a corrective action advisory message.

I3. This module must allow users the capability to update the display when an aircraft changes partition.

I4. This module must allow users to update the display when a potential threat transitions into a collision warning situation.

Local Dictionary

!color_type!   enumerated (red, orange, green, yellow, white, blue, black, pink, purple, indigo, violet)

!cws_id!   Enumerated name of the collision warning situation.

!host_color!   record of (
    cws_name : !cws_id!,
    color : !color_type!
)

3.1. Corrective_Action_Message

Concrete Operations

Operation	Parameter	Description	Undesired Events
corrective_action_msg	threat:Potential_threat.pt_handle;I		None

Output Produced

Item	Type	Operation
msg	string	Air_Traffic_Display_Device.write_text
xloc	Air_Traffic_Display_Device.position_type	
yloc	Air_Traffic_Display_Device.position_type	

Effects

corrective_action_msg   Determines the content of the Corrective_Action_Msg and sends it to the Air_Traffic_Display_Device.

Function Definition

Output Values:	msg: !corrective_message! xloc : 370 yloc: 980

Local Dictionary

!corrective_message!   The following table defines the content of the corrective_action message based upon initial conditions (given in the topmost row of the table) and conditions that hold when the host aircraft and potential threat are closest together (leftmost column). The following abbreviations are used in the table.

alt_H host aircraft.altitude
alt_{PT} potential threat.altitude
$rate_H$ host aircraft.climb_rate
$rate_{PT}$ potential threat.climb_rate

	$alt_H \geq alt_{PT}$ AND $alt_H - alt_{PT} \geq 500$ feet	$alt_H \geq alt_{PT}$ AND $alt_H - alt_{PT} < 500$ feet
$msd \geq 500$ feet	maintain current heading and rate	maintain current heading and rate
$msd < 500$ feet AND $rate_H = 0$ AND $rate_{PT} = 0$	————————————————	climb at X ft/min
$msd < 500$ feet AND $rate_H = 0$ AND $rate_{PT} < 0$	————————————————	climb at X ft/min
$msd < 500$ feet AND $rate_H = 0$ AND $rate_{PT} > 0$	climb at X ft/min	climb at X ft/min
$msd < 500$ feet AND $rate_H < 0$ AND $rate_{PT} = 0$	fly level	climb at X ft/min
$msd < 500$ feet AND $rate_H < 0$ AND $rate_{PT} < 0$	fly level	climb at X ft/min
$msd < 500$ feet AND $rate_H < 0$ AND $rate_{PT} > 0$	climb at X ft/min	climb at X ft/min
$msd < 500$ feet AND $rate_H > 0$ AND $rate_{PT} = 0$	maintain current heading and rate	climb at X ft/min
$msd < 500$ feet AND $rate_H > 0$ AND $rate_{PT} < 0$	————————————————	climb at X ft/min
$msd < 500$ feet AND $rate_H > 0$ AND $rate_{PT} > 0$	climb at X ft/min	climb at X ft/min

	$alt_H < alt_{PT}$ AND $alt_{PT} - alt_H \geq 500$ feet	$alt_H < alt_{PT}$ AND $alt_{PT} - alt_H < 500$ feet
msd $\geq$ 500 feet	maintain current heading and rate	maintain current heading and rate
msd < 500 feet AND $rate_H = 0$ AND $rate_{PT} = 0$	--	dive at X ft/min
msd < 500 feet AND $rate_H = 0$ AND $rate_{PT} < 0$	dive at X ft/min	dive at X ft/min
msd < 500 feet AND $rate_H = 0$ AND $rate_{PT} > 0$	--	dive at X ft/min
msd < 500 feet AND $rate_H < 0$ AND $rate_{PT} = 0$	maintain current heading and rate	dive at X ft/min
msd < 500 feet AND $rate_H < 0$ AND $rate_{PT} < 0$	dive at X ft/min	dive at X ft/min
msd < 500 feet AND $rate_H < 0$ AND $rate_{PT} > 0$	--	dive at X ft/min
msd < 500 feet AND $rate_H > 0$ AND $rate_{PT} = 0$	fly level	dive at X ft/min
msd < 500 feet AND $rate_H > 0$ AND $rate_{PT} < 0$	fly level	dive at X ft/min
msd < 500 feet AND $rate_H > 0$ AND $rate_{PT} > 0$	fly level	dive at X ft/min

where the quantity "X" appearing in the preceding text messages is computed as

$$X = (500 - msd) / t_{msd}$$

Item t_{msd} is the time to intersect. Entries marked with dashed lines denote conditions not physically possible.:

3.2. Update_Aircraft_Display_Symbol

Concrete Operations

Operation	Parameter	Description	Undesired Events
update_ads	threat:Potential_Threat.pt_handle;I		None

Output Produced

Item	Type	Operation
id shape	Air_Traffic_Display_Device.display_handle Air_Traffic_Display_Device.shape_type	Air_Traffic_Display_Device.shape_object

Effects

update_ads   Updates the icon shape for the specified potential threat when its partition changes.

Function Definition

	Condition	
	partition_threat.partition = ID	partition_threat.partition = UID
Output Value:	id shape : id_shape	id shape = uid_shape

Local Dictionary

!icon_shape!   record of (
                     host_shape : identifier,
                     id_shape : identifier,
                     uid_shape : identifier
)

3.3. Update_ATD

Concrete Operations

Operation	Parameter	Description	Undesired Events
update_cws	threat:Potential_Threat.pt_handle;I display_status: Potential_Threat.target_display;I		None

Output Produced

NOTE: The output produced varies as a function of the value of parameter display_status and whether the target designated by parameter threat has been displayed. There are three cases described below. Only one of these outputs is produced per invocation of routine "update_cws".

Case 1: display_status = delete

Item	Type	Operation
handle	Air_Traffic_Display_Device.display_handle	Air_Traffic_Display_Device.delete_object

Case 2: threat has not been displayed yet

Item	Type	Operation
icon_shape	Air_Traffic_Display_Device.shape	Air_Traffic_Display_Device.create_object
icon_size	Air_Traffic_Display_Device.size	
icon_fill	Air_Traffic_Display_Device.fill	
icon_color	Air_Traffic_Display_Device.color	
fill_blink_rate	Air_Traffic_Display_Device.blink	
obj_blink_rate	Air_Traffic_Display_Device.blink	
xloc	Air_Traffic_Display_Device.position	
yloc	Air_Traffic_Display_Device.position	
label_1	string	
label_2	string	
label_3	string	
label_4	string	
label_5	string	

Case 3: default case

Item	Type	Operation
shape	Air_Traffic_Display_Device.shape	Air_Traffic_Display_Device.move_object
size	Air_Traffic_Display_Device.size	
fill	Air_Traffic_Display_Device.fill	
color	Air_Traffic_Display_Device.color	
fill_blink_rate	Air_Traffic_Display_Device.blink	
obj_blink_rate	Air_Traffic_Display_Device.blink	
xloc	Air_Traffic_Display_Device.position	
yloc	Air_Traffic_Display_Device.position	
label_1	string	
label_2	string	
label_3	string	
label_4	string	
label_5	string	

endif

Function Definition

Event
@T(!radar_msg!)

Output Value(s):	potential_threat:
	    < forall X in display >
	        < if X.CWS_Def[potential_threat] then >
	           id:
	           < if X.Partition = ID then >
	           shape: < icon_shape.id >
	           < else >
	           shape : < icon_shape.uid >
	           < endif >
	           size: 16
	           < if X.PT_Fill then >
	           fill: < X.PT_Color >
	           < else >
	           fill: none
	           < endif >
	           color: < X.PT_Color >
	           < if X.PT_Blink then >
	           fill_blink_rate: 0.125
	           obj_blink_rate: 0.125
	           < else >
	           fill_blink_rate: 0
	           obj_blink_rate : 0
	           < endif >
	           (xloc, yloc) : !!location!!
	           label_1: "ID: " + !potential_threat_ID!
	           label_2: "Altitude: " + !potential_threat_altitude!
	           label_3: "Airspeed: " + !potential_threat_airspeed!
	           label_4: "Course: " + !potential_threat_course!
	           label_5: "Range: " + !potential_threat_range!
	        < endif >
	    < endfor >
	
	host:
	    id:
	    shape: < icon_shape.host_shape >
	    size: 16
	    fill: none
	    color: !!host_color!!
	    fill_blink_rate: 0
	    obj_blink_rate : 0
	    xloc: (1270 / 2 - 16 / 2)
	    yloc: (1000 / 2 - 16 / 2)
	    label_1: " "
	    label_2: " "
	    label_3: " "
	    label_4: " "
	    label_5: " "

Local Dictionary

!!host_color!! | The value returned by an internal program get_host_color.

!!location!! | The (x,y) location value returned by an internal program calculate_location.

!arithmetic_opr! | enumerated (lt, le) which are arithmetic operators **less than** and **less than or equal to**, respectively.

!constant! | numeric quantity

!cws_id! | Enumerated name of the collision warning situation.

!cws_predicate! |

```
union of  (
     expr :  record of  (    op1 : !cws_predicate!,
                             op2 : !cws_predicate!,
                             opr : OR),
     time :  record of  (    op1 : !constant!,
                             opr : !arithmetic_opr!),
     range : record of  (    op1 : !constant!,
                             opr : !arithmetic_opr!)
)
```

!potential_threat_airspeed! | An ASCII representation of the airspeed of a given potential threat.

!potential_threat_altitude! | An ASCII representation of the altitude of a given potential threat.

!potential_threat_course! | An ASCII representation of the ground track (course) of a given potential threat.

!potential_threat_ID! | An ASCII representation of the identification number of a given potential threat.

!potential_threat_range! | An ASCII representation of the range between the host aircraft and a given potential threat.

!pt_display!

```
record of  (
    cws_name : !cws_id!,
    cws_def : !cws_predicate!,
    partition : enumerated (ID, UID, ALL)
    color :  identifier,
    blink : boolean,
    fill : boolean
)
```

!radar_msg!

The event that occurs when another Radar_Msg has been received from the radar device.

3.4. Initialize_Display

Concrete Operations

Operation	Parameter	Description	Undesired Events
initialize_display	———————————————————————		None

Output Produced

Item	Type	Operation
xloc_c	Air_Traffic_Display_Device.position	Air_Traffic_Display_Device.create_display
yloc_c	Air_Traffic_Display_Device.position	
width	Air_Traffic_Display_Device.position	
height	Air_Traffic_Display_Device.position	
icon_shape	Air_Traffic_Display_Device.shape	Air_Traffic_Display_Device.create_object
icon_size	Air_Traffic_Display_Device.size	
icon_fill	Air_Traffic_Display_Device.fill	
icon_color	Air_Traffic_Display_Device.color	
fill_blink_rate	Air_Traffic_Display_Device.blink	
obj_blink_rate	Air_Traffic_Display_Device.blink	
xloc	Air_Traffic_Display_Device.position	
yloc	Air_Traffic_Display_Device.position	
label_1	string	
label_2	string	
label_3	string	
label_4	string	
label_5	string	

Effects

initialize_display

A window is created on the display. The window location is given by (xloc, yloc) and its size is specified by width (horizontal length) and height (vertical length). In addition, an icon for the host aircraft is created and positioned in the center of the window. The host aircraft icon has the following initial characteristics:

icon_shape:	< icon_shape.host_shape >
icon_size:	16
icon_fill :	black
icon_color:	*color for normal cws_status*
fill_blink_rate:	0.0
obj_blink_rate:	0.0
xloc:	
yloc:	
label_1:	" "
label_2:	" "
label_3:	" "
label_4:	" "
label_5:	" "

Function Definition

Output Values:	xloc_c : 0 yloc_c : 0 width : 1270 height : 1000
	icon_shape : < icon_shape.host_shape > icon_size : 16 icon_fill : black icon_color : white fill_blink_rate : 0.0 obj_blink_rate : 0.0 xloc : 1270/2 – 16/2 yloc : 1000/2 – 16/2 label_1 : " " label_2 : " " label_3 : " " label_4 : " " label_5 : " "

4. Air_Traffic_Display_Device (ATDD)

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Interface Requirement

I1. This module must allow aircraft status information to be displayed on the ATD.

I2. This module must allow users to initialize the display.

Concrete Operations

Operation	Parameter	Description	Undesired Events
write_text	msg:string;I xloc:position;I yloc:position;I		None
create_object	icon_shape:shape;I icon_size:size;I icon_fill:fill;I icon_color:color;I fill_blink_rate:blink;I obj_blink_rate:blink;I xloc:position;I yloc:position;I label_1:string;I label_2:string;I label_3:string;I label_4:string;I label_5:string;I id:display_handle;O		None
delete_object	id:display_handle;I		None
move_object	id:display_handle;I xloc:position;I yloc:position;I label_1:string;I label_2:string;I label_3:string;I label_4:string;I label_5:string;I		None
chg_object_blink	id:display_handle;I fill_blink_rate:blink;I obj_blink_rate:blink;I		None
chg_object_fill	id:display_handle;I icon_fill:fill;I		None
chg_object_color	id:display_handle;I icon_color:color;I		None

Operation	Parameter	Description	Undesired Events
chg_object_shape	id:display_handle;I icon_shape:shape;I		None
create_display	xloc:position;I yloc:position;I width:position;I height:position;I		None

Effects

chg_object_blink	The specified object is made to blink at the specified rate.
chg_object_color	The color of the object is changed.
chg_object_fill	Causes the icon interic to be filled in with the specified color.
chg_object_shape	Changes the icon shape of the specified object.
create_display	Creates a display window having origin (xloc, yloc) with the specified height and width. Only one display window can be opened at a time.
create_object	Creates a new object on the display with the specified attributes and labels. A unique identifier for the object is returned to the calling program.
delete_object	The specified object is removed from the display.
move_object	Moves the specified object to the new (xloc, yloc) location.
write_text	The text message **msg** is displayed on the ATD at location (xloc, yloc). The previous message displayed message is overwritten.

Events Signalled – none

Types

blink   The blinking rate for a displayed icon. Data
                                       representation:

Minimum value:	0.0
Maximum value:	10.0
Units: seconds	
Resolution:	0.1 second

                                       Value 0.0 means do not blink.

color   enumerated (none, red, orange, green, yellow, white,
                                       blue, black, pink, purple, indigo, violet)

fill   Same as color.

display_handle   A unique identifier for a particular display object.

position   Pixel number. Data representation:

Minimum value:	0
Maximum value:	1,100
Units: pixels	
Resolution:	1 pixel

shape   enumerated (square, circle, triangle)

size   Size of icon in pixels. Data representation:

Minimum value:	1
Maximum value:	100
Units: pixels	
Resolution:	1 pixel

Local Dictionary

Undesired Event Dictionary – none

5. Audible_Alarm (AA)

Instantiation Parameters

Parameter	Type	Description
ring	list of !ring_info!	Each record defines the frequency and duration to initiate the audible alarm for an aircraft in a specified collision warning situation.

Instantiation Constraints

1.   Adaptable component Audible_Alarm_Device must be instantiated as well.

Assumptions

   Dependency Assumption

D1. There is a way to initiate the audible alarm at a specific frequency and time duration.

Interface Requirement

I1. This module must allow users to determine what frequency and duration to ring the audible alarm. This module must allow users to ring the audible alarm.

Concrete Operations

Operation	Parameter	Description	Undesired Events
ring_alarm	cws:Potential_Threat.cws_id;I	!cws_id!	None

Output Produced

Item	Type	Operation
frequency duration	Audible_Alarm_Device.frequency Audible_Alarm_Device.duration	Audible_Alarm_Device.ring_alarm

Events Signalled – none

Types – none

Local Dictionary

!cws_id!   Enumerated name of the collision warning situation.

!ring_info!   record of   (
        cws_name : !cws_id!.
        frequency : integer.
        duration : real
)

6. Audible_Alarm_Device (AAD)

The audible alarm device generates a tone that can be heard within the host aircraft cockpit.

Instantiation Parameters

Parameter	Type	Description
loosely-coupled	boolean	A value of **true** indicates that the communication between the ring_alarm and the calling operation should be loosely-coupled; **false** means they should be tightly coupled.

Instantiation Constraints

1. If loosely-coupled is **true**, then adaptable component module Temporary_Data_Buffer must be instantiated as well.

Assumptions

Interface Requirement

I1.   This module must allow the audible alarm to be rung at a specified frequency and time duration.

Concrete Operations

Operation	Parameter	Description	Undesired Events
ring_alarm	f:frequency;I d:duration;I	!alarm_frequency! !alarm_duration!	None

Effects

ring_alarm   The audible alarm is rung at frequency **f** for a time duration **d**.

Events Signalled – none

Types

duration   Length of time that the audible alarm is rung. Data *representation:*

Minimum value:	0.01
Maximum value:	10.00
Units: seconds	
Resolution:	0.01 seconds

frequency   Pitch of the tone made by the audible_alarm. Data representation:

Minimum value:	1,000
Maximum value:	10,000
Units: hertz	
Resolution:	1 hertz

Local Dictionary

!alarm_duration!   Specifies the number of seconds to ring the audible alarm.

!alarm_frequency!   Specifies the frequency, in hertz, at which to ring the audible alarm.

Undesired Event Dictionary – none

7. Collision_Warning_Situation_Status (CWSS)

This module determines the collision warning situation status for potential threats and the host aircraft.

Instantiation Parameters

Parameter	Type	Description
cws	list of !cws_info!	Each record in this list contains the name of the collision warning situation and a boolean-valued expression that defines the criteria for the named situation.

Instantiation Constraints

1. The cws parameter cannot be empty.

2. The records in cws must be ordered in decreasing severity levels before instantiating this module.

Assumptions

Dependency Assumption

D1. There is a way to predict how much time elapses before two aircraft reach a separation minima.

D2. There is a way to determine the potential threat range.

D3. There is a way to determine what the previous collision warning situation status was for the specified potential threat.

Interface Requirement

I1. This module must allow users to determine the collision warning situation status of a potential threat.

I2. This module must allow users to determine the collision warning situation status of the host aircraft.

Concrete Operations

Operation	Parameter	Description	Undesired Events
determine_cws_status	threat:Potential_Threat.pt_handle;I cws:Potential_Threat.cws_id;O		None
determine_host_cws_status	cws:Potential_Threat.cws_id;O		None

Effects

determine_cws_status	Determines the collision warning situation status for the specified potential threat.
determine_host_cws_status	Determines the collision warning situation status for the host aircraft.

Events Signalled – none

Types – none

Local Dictionary

!constant!   integer numeric quantity

!cws_info!   record of (
                                            cws_name : identifier,
                                          severity : integer,
                                          predicate : !cws_predicate!,
                                          partition : enumerated (ID, UID, ALL)
)

!cws_predicate!   union of (
                                            time : record of (min : !constant!,
                                                                  max : !constant!),
                                          range : record of (min : !constant!,
                                                                    max : !constant!),
                                          time_and_range : record of (
                                              (t_min : !constant!,
                                                  t_max : !constant!,
                                                  r_min : !constant!,
                                                  r_max : !constant!
)
)

Undesired Event Dictionary – none

8. Communication (COMM)

Parameter	Type	Description
atc_msg	list of !atc_info!	Each record indicates the transponder code for the ATC_Msg sent to the air traffic control center in response to a specific collision warning situation.
inter_air_msg	list of !inter_air_info!	Each record indicates the transponder code for the Inter_Air_Msg sent to the air traffic control center in response to a specific collision warning situation.
mode	!mode!	Message content for the ATC_Msg.

Instantiation Constraints

1. If atc_msg is nonempty, then parameter atc_msg of adaptable component Communication_Device must be **true**.

2. If inter_air_msg is nonempty, then parameter inter_air_msg of adaptable component Communication_Device must be **true**.

3. Adaptable component Communication_Device must be instantiated as well.

Assumptions

Dependency Assumption

D1. There is a way to determine which aircraft has triggered the event requiring the need to generate an ATC_Msg.

D2. There is a way to determine which aircraft has triggered the event requiring the need to generate an Inter_Air_Msg.

D3. There is a way to determine the collision warning situation the potential threat is in relative to the host aircraft.

D4. There is a way of obtaining the altitude of the host aircraft.

D5. There is a way of obtaining the latitude and longitude of the host aircraft.

D6. There is a way to send the ATC_Msg to the air traffic control center.

D7. There is a way to send the Inter_Air_Msg to the appropriate potential threat.

Interface Requirement

I1. This module must allow users to format and transmit an ATC_Msg or Inter_Air_Msg.

< if there exists C ∈ atc_msg then >

8.1. ATC_Msg

Concrete Operations

Operation	Parameter	Description	Undesired Events
send_atc_msg	cws:Potential_Threat.cws_id;I	!cws_id!	None

Output Produced

< if mode = Mode_A then >

Item	Type	Operation
atc_msg: code	positive	Communication_Device.send_atc_msg

< else >

Item	Type	Operation
atc_msg: code altitude	positive Physical_Quantities.feet	Communication_Device.send_atc_msg

< endif >

Local Dictionary

!atc_info!	record of (cws_name : !cws_id!, code : positive)
!cws_id!	Enumerated name of the collision warning situation.
!mode!	enumerated (Mode_A, Mode_C)

< endif >

< if there exists $X \in$ inter_air_msg then >

8.2. Inter_Air_Msg

Concrete Operations

Operation	Parameter	Description	Undesired Events
send_ia_msg	cws:Potential_Threat.cws_id:I	!cws_id!	None

Output Produced

Item	Type	Operation
inter_air_msg: code altitude latitude longitude	positive Physical_Quantities.feet Physical_Quantities.degrees Physical_Quantities.degrees	Communication_Device.send_inter_air_msg

Local Dictionary

!cws_id!	Enumerated name of the collision warning situation.

!inter_air_info!	record of (
	      cws_name : !cws_id!,
	      code : positive
	)

< endif >

9. Communication_Device (CD)

Instantiation Parameters

Parameter	Type	Description
atc_msg	boolean	A value of **true** indicates that functionality must be included to enable transmission of an ATC_Msg to an air traffic control center. Otherwise, this parameter's value is **false**.
inter_air_msg	boolean	A value of **true** indicates that functionality must be included to enable transmission of an Inter-Air_Msg to a potential threat. Otherwise, this parameter's value is **false**.
mode	!mode!	Message type for the ATC_Msg. This parameter is omitted when atc_msg is **false**.
loosely-coupled	boolean	A value of **true** indicates that the communication between either send_atc_msg or send_inter_air_msg and the calling operation should be loosely-coupled; **false** means they should be tightly coupled.

Instantiation Constraints

1. If parameter loosely-coupled is **true**, then adaptable component Temporary_Data_Buffer must be instantiated as well.

Assumptions

Interface Requirement

I1. This module must allow the messages to be sent to either the nearest air traffic control center or to the appropriate potential threat.

Concrete Operations

Operation	Parameter	Description	Undesired Events

< if atc_msg then >

Operation	Parameter	Description	Undesired Events
send_atc_msg	code: natural;I **< if mode = C then >** altitude:Physical_Quantities.feet;I **< endif >**		None

< endif >

< if inter_air_msg then >

Operation	Parameter	Description	Undesired Events
send_inter_air_msg	code: natural;I altitude:Physical_Quantities.feet;I latitude:Physical_Quantities.degrees;I longitude:Physical_Quantities.degrees;I		None

< endif >

Effects

< if atc_msg then >

send_atc_msg   The ATC_Msg is sent to nearest air traffic control center.

< endif >

< if inter_air_msg then >

send_inter_air_msg   The Inter_Air_Msg is sent to the appropriate potential threat.

< endif >

Events Signalled – none

Types – none

Local Dictionary

!mode!   enumerated (A. C)

Undesired Event Dictionary – none

10. Extended_Computer (EC)

This module provides an integrated abstraction of processor. operating system. and language capabilities. The *Reference Manual for the Ada Programming Language* (United States Department of Defense 1983) provides the abstract interface for this module.

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Interface Requirement

I1. This module must allow users to convert an integer quantity into its equivalent string image.

11. Host_Aircraft (HA)

This module models the host aircraft for an ATD/CWM system. The host aircraft has properties of altitude. airspeed. location, bearing, and climb rate.

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Dependency Assumption

D1. There is a way to obtain altitude, velocity, location, and ground_track for the host aircraft.

D2. There is a way to determine the collision warning situation status of the host aircraft.

D3. There is a way to determine the climb rate of the host aircraft.

Interface Requirement

I1. This module must allow users to reference properties of the host aircraft.

Concrete Operations

Operation	Parameter	Description	Undesired Events
get_altitude	altitude:Physical_Quantities.feet:O	!altitude!	None
get_climb_rate	rate:Physical_Quantities.fpm:O	!rate!	None
get_cws_status	status:Potential_Threat.cws_id:O		None
get_ground_track	ground_track: Physical_Quantities.degrees:O	!ground_track!	None
get_host_data	altitude:Physical_Quantities.feet:O ground_track: Physical_Quantities.degrees:O rate:Physical_Quantities.fps;O airspeed:Physical_Quantities.knots:O latitude:Physical_Quantities.latitude;O longitude:Physical_Quantities.longitude:O status:Potential_Threat.cws_id:O	!altitude! !ground_track! !rate! !airspeed! !location!	None
get_location	latitude:Physical_Quantities.latitude:O longitude:Physical_Quantities.longitude:O	!location!	None
get_velocity	airspeed:Physical_Quantities.knots:O	!velocity!	None

Effects

get_altitude   Returns the most recently measured altitude of the host aircraft.

get_climb_rate   Returns the most recently measured climb rate of the host aircraft.

get_cws_status   Returns the collision warning situation status of the host aircraft.

get_ground_track	Returns the most recently measured ground_track of the host aircraft.
get_host_data	This function returns the values for all properties of the host aircraft.
get_location	Returns the most recently measured position of the host aircraft.
get_velocity	Returns the most recently measured velocity of the host aircraft.

Events Signalled – none

Types – none

Local Dictionary

!altitude!	The vertical distance height of the host aircraft measured from mean sea level.
!ground_track!	Ground_track of the host aircraft measured from the line of the host aircraft to magnetic north to the horizontal component of the host aircraft's x axis in the clockwise direction looking down.
!location!	Location given in terms of latitude and longitude.
!rate!	Climb rate of the host aircraft.
!velocity!	Velocity of the host aircraft.

Undesired Event Dictionary – none

12. Initialization_and_Termination (IT)

The hidden decision of this module is how ATD/CWM system operation is initiated and terminated.

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Dependency Assumption

D1. There is a way to initialize the air traffic display.

Concrete Operations – none

Effects

Events Signalled – none

Types – none

Local Dictionary

Undesired Event Dictionary – none

13. Navigation (NAV)

The navigation device reports host aircraft flight characteristics altitude, velocity, ground_track, latitude, and longitude to the host aircraft.

Instantiation Parameters – none

Instantiation Constraints

Assumptions

Interface Requirement

I1. This module must provide altitude, velocity, latitude, longitude, and ground_track for the host aircraft.

Concrete Operations

Operation	Parameter	Description	Undesired Events
get_nav_data	altitude:Physical_Quantities.feet:O timestamp : Physical_Quantities.seconds;O airspeed:Physical_Quantities.knots:O ground_track: Physical_Quantities.degrees;O latitude:Physical_Quantities.latitude;O longitude:Physical_Quantities.longitude;O		None

Effects

Events Signalled – none

Types – none

Local Dictionary

Undesired Event Dictionary – none

14. Numerical_Algorithms (NA)

This module provides the mathematical service routines that are required by more than one module in the system. Most values are a predefined arithmetic function of the input parameters.

Instantiation Parameters – none

Instantiation Constraints

Assumptions

Interface Requirement

I1. This module must provide an operation for computing the square root of an real quantity.

I2. This module must provide trignometric operations.

Concrete Operations

Operation	Parameter	Description	Undesired Events
arccos	p1:real:I p2:real:O		domain_error
cos	p1:real:I p2:real:O		None
sin	p1:real:I p2:real:O		None
sqrt	p1:real:I p2:real:O	!sqrt!	root_negative

Effects

arccos   Inverse trignometric cosine function. Produces in p2 the angle whose cosine is p1.

cos   Produces in p2 the cosine of angle p1.

sin   Produces in p2 the sine of angle p1.

sqrt   Returns the square root of p1 in p2, if p1 is non-negative.

Events Signalled – none

Types – none

Local Dictionary

!sqrt!   The square root of p1, if p1 is non-negative.

Undesired Event Dictionary

domain_error   The magnitude of argument p1 is greater than 1.

root_negative   Argument p1 of operation sqrt is negative.

15. Physical_Quantities (PQ)

This module implements data types needed for representing and calculating physical quantities.

Instantiation Parameters – none

Instantiation Constraints

Assumptions

Dependency Assumption

D1.   There is a way to obtain the current time of day.

Interface Requirement

I1.   This module must provide an operation to convert degrees to radians.

I2.   This module must provide an operation to convert radians to degrees.

I3.   This module must provide operations for comparing values of the same data type.

I4.   This module must provide operations for assigning values of the same data type.

I5.   This module must provide operations for converting two values of the same data type into different representations.

I6.   This module must provide an operation to return the current time of day expressed as elapsed seconds since midnight.

Concrete Operations

Operation	Parameter	Description	Undesired Events
get_time	current_time:seconds:O	!time!	None

Effects

Events Signalled – none

Types

degrees   Angles measured in degrees. Data representation:
                                 Minimum value: –360.0
                                 Maximum value: 360.0
                                 Units: degrees
                                 Resolution:0.1 degrees

feet   Distance measured in feet. Data representation:
                                 Minimum value: –200.0
                                 Maximum value: 350,000.0
                                 Units: foot
                                 Resolution:1 foot

fpm
Velocity measured in feet per minute. Data representation:
    Minimum value: -76,000
    Maximum value: 76,000
    Units: feet per minute
    Resolution: 1 foot per minute

fps
Velocity measured in feet per second. Data representation:
    Minimum value: -1,250
    Maximum value: 1,250
    Units: feet per second
    Resolution: 1 foot per second

knots
Velocity measured in nautical miles per hour. Data representation:
    Minimum value: 0
    Maximum value: 750
    Units: nautical miles per hour
    Resolution: 1 nautical mile per hour

latitude
The angular distance north or south of the equator measured in degrees. A negative value indicates latitude south of the equator; a positive value is latitude north of the equator. Data representation:
    Minimum value :   -90.0
    Maximum value :   90.0
    Units : degrees
    Resolution : 0.1 degree

longitude
The angular distance on the earth east or west of the prime meridian at Greenwich, England, to the point on the earth's surface for which the longitude is being determined, expressed in degrees. A negative value is longitude east of the prime meridian; a positive value is longitude west of the prime meridian. Data representation:
    Minimum value :   -180.0
    Maximum value :   180.0
    Units : degrees
    Resolution : 0.1 degree

nautical_mile

Distance measured in nautical miles. Data representation:
    Minimum value: −3,000.0
    Maximum value: 3,000.0
    Units: nautical_mile
    Resolution:0.1 nautical_mile

radians

Angles measured in radians. Data representation:
    Minimum value: −10.000
    Maximum value: 10.000
    Units: radians
    Resolution: 0.001 radian

seconds

Time measured in seconds. Data representation:
    Minimum value: 0
    Maximum value: 20,000.000
    Resolution:0.1 second

Local Dictionary

!time!

Current time in seconds on a 24-hour clock. Values returned are in the range 0.0 to 86,400.0 seconds, inclusive, where 0.0 represents midnight.

Undesired Event Dictionary − none

16. Potential_Threat (PT)

Instantiation Parameters

Parameter	Type	Description
cws	list of !cws_info!	Each record defines a collision warning situation and its corresponding responses.

Instantiation Constraints − none

Assumptions

Dependency Assumption

D1.   There is a way to obtain altitude, aircraft_identification, velocity, range, ground_track, and relative_bearing for a potential threat.

D2.   There is a way to determine which partition a potential threat belongs to.

D3.   There is a way to determine the collision warning situation status of the potential threat.

D4. There is a way to format and transmit a corrective_action advisory message.

D5. There is a way to determine what frequency and duration to ring the audible alarm.

D6. There is a way to format and transmit an ATC_Msg or Inter_Air_Msg.

D7. There is a way to update the display when the collision warning situation status of a potential threat changes.

Interface Requirement

I1. This module must allow users to reference properties of a potential threat.

I2. This module must allow users to determine the potential threat partition.

Concrete Operations

Operation	Parameter	Description	Undesired Events
altitude_valid	threat:pt_handle;I altitude_status:boolean;O		None
get_aircraft_id	threat:pt_handle;I aircraft_id:string(8);O		None
get_altitude	threat:pt_handle;I altitude:Physical_Quantities.feet;O	!altitude!	None
get_climb_rate	threat:pt_handle;I rate:Physical_Quantities.fpm;O	!rate!	None
get_cws_status	threat:pt_handle;I cws_status:cws_id;O		None
get_ground_track	threat:pt_handle;I ground_track: Physical_Quantities.degrees:O	!ground_track!	None
get_partition	threat:pt_handle;I partition:partition;O		None
get_range	threat:pt_handle;I range: Physical_Quantities.nautical_mile;O	!range!	None
get_relative_bearing	threat:pt_handle;I relative_bearing: Physical_Quantities.degrees;O	!relative_bearing!	None
get_velocity	threat:pt_handle;I velocity:Physical_Quantities.knots;O	!velocity!	None
velocity_valid	threat:pt_handle;I status:boolean;O		None

Effects

altitude_valid

Returns a status indicating whether the most recent altitude value for the specified aircraft is valid. **True** means the altitude is valid: **false** means that it is invalid.

get_aircraft_id

Returns the aircraft_identification of the specified potential threat.

get_altitude

Returns the current altitude of the specified potential threat.

get_climb_rate

Returns the most recently measured vertical rate of change of the specified potential threat.

get_cws_status

Returns the most recent collision warning situation status of the specified potential threat.

get_ground_track

Returns the current ground_track of the specified potential threat.

get_partition

Returns which partition the potential threat is a member of.

get_range

Returns the most recently measured range of the specified potential threat.

get_relative_bearing

Returns the most recently measured relative_bearing of the specified potential threat.

get_velocity

Returns the most recently measured velocity of the specified potential threat. Undesired event **velocity_invalid** is raised if the velocity value for the specified potential threat is invalid.

velocity_valid

Returns a status indicating whether the velocity value for the specified aircraft is valid. **True** means the velocity is valid: **false** means that it is invalid.

Function Definition

< forall C in cws >

	Event

	< if C.partition = ALL then >
	    @T(< C.cws_def >[potential_threat])
	< else >
	    @T(< C.cws_def >[potential_threat]) when partition = < C.partition >
	< endif >

Output Value(s):	< if C.alarm then >
	    Audible_Alarm.ring_alarm(< C.cws_name >)
	< endif >
	< if C.atc_msg then >
	    Communication.send_atc_msg(< C.cws_name >)
	< endif >
	< if C.inter_air_msg then >
	    Communication.send_ia_msg(< C.cws_name >)
	< endif >
	< if C.corrective then >
	    Air_Traffic_Display.corrective_action_msg(potential_threat)
	< endif >
	Air_Traffic_Display.update_cws(potential_threat)

< endfor >

	Event
	@T(< ADS.ID_Shape.Partition >[potential_threat])
Output Value:	Air_Traffic_Display.update_ads(potential_threat)

	Event
	@F(< ADS.ID_Shape.Partition >[potential_threat])
Output Value:	Air_Traffic_Display.update_ads(potential_threat)

Events Signalled – none

Types

cws_id	enumerated (< forall C in cws > < C.cws_name >, < endfor > NORMAL)
partition	enumerated (ID, UID)
pt_handle	A unique identifier for a particular potential threat.

Local Dictionary

!altitude!

The vertical distance height of the potential threat measured from mean sea level.

!constant!

numeric quantity

!cws_info!

```
record of (
    cws_name : identifier.
    severity : integer,
    predicate : !cws_predicate!,
    partition : enumerated (ID, UID, ALL),
    alarm : boolean,
    atc_msg : boolean,
    inter_air_msg : boolean,
    corrective : boolean
)
```

!cws_predicate!

```
union of   (
    time :  record of  (    min : !constant!,
                            max : !constant!),
    range : record of  (    min : !constant!,
                            max : !constant!),
    time_and_range : record of (
                    (   t_min : !constant!,
                        t_max : !constant!,
                        r_min : !constant!,
                        r_max : !constant!
                    )
)
```

!ground_track!

Ground_track of the potential threat measured from the line of the potential threat to magnetic north to the horizontal component of the potential threat's x axis in the clockwise direction looking down.

!range!

Distance from the potential threat to the host aircraft.

!rate!

Climb rate of the potential threat.

!relative_bearing!

Bearing of the potential threat relative to the host aircraft. Relative_bearing is measured from the ground track of the host aircraft to the line from the host aircraft to the potential threat in the clockwise direction looking down.

!velocity!

Velocity of the potential threat.

Undesired Event Dictionary – none

17. Potential_Threat_Partition (PTP)

This module knows how to determine the potential threat partition.

Instantiation Parameters

Parameter	Type	Description
altitude	enumerated (True, False)	A value of **True** means that the altitude must be known in order for the potential threat to be considered **identified**. Otherwise, this parameter is **False**.
airspeed	enumerated (True, False)	A value of **True** means that the airspeed must be known in order for the potential threat to be considered **identified**. Otherwise, this parameter is **False**.

Instantiation Constraints

1. Either parameter altitude, airspeed, or both must be **True**.

Assumptions

Dependency Assumption

D1. There is a way to determine the availability of a potential threat's altitude or airspeed.

Interface Requirement

I1. This module must determine the partition of which a potential threat is a member.

Concrete Operations

Operation	Parameter	Description	Undesired Events
get_partition	threat:Potential_Threat.pt_handle:I partition:Potential_Threat.partition:O		None

Effects

get_partition   Returns the partition of which the potential threat is a member.

Events Signalled – none

Types – none

Local Dictionary

Undesired Event Dictionary – none

18. Radar (RADAR)

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Interface Requirement

I1.   This module must provide aircraft_identification, range, and relative_bearing for a potential threat.

Concrete Operations

Operation	Parameter	Description	Undesired Events

get_radar_data	aircraft_id:string(8):()
	sweep:integer:()
	relative_bearing:
	    Physical_Quantities.degrees:()
	range:
	    Physical_Quantities.nautical_mile:()
	timestamp:Physical_Quantities.seconds:()
			None

Effects

Events Signalled – none

Types – none

Local Dictionary

Undesired Event Dictionary – none

19. Situation_Dynamics (SD)

Instantiation Parameters – none

Instantiation Constraints – none

Assumptions

Dependency Assumption

D1.   There is a way to determine velocity, climb rate, altitude, and ground_track of the host aircraft.

D2.   There is a way to compute trigonometric functions.

D3.   There is a way to compute the square root.

D4.   There is a way to determine the horizontal component of an aircraft's velocity.

D5. There is a way to determine the range component that lies in the X-Y plane.

D6. There is a way to determine the velocity, climb rate, altitude, ground_track, relative_bearing, and range of the potential threat.

Interface Requirement

I1. This module must determine how much time elapses before two aircraft reach a separation minima.

I2. This module must determine the separation minima two aircraft will pass within each other.

Concrete Operations

Operation	Parameter	Description	Undesired Events
get_elapsed_time	threat:Potential_Threat.pt_handle:I time:Physical_Quantities.seconds:O	!elapsed_time!	None
get_msd	threat:Potential_Threat.pt_handle:I distance:Physical_Quantities.feet:O	!minimal!	

Effects

get_elapsed_time

Returns the predicted elapsed time before the host aircraft and specified potential threat reach the predicted closest range.

get_msd

Returns the predicted closest range between the host aircraft and specified potential threat assuming no changes in their respective flight characteristics.

Events Signalled – none

Types – none

Local Dictionary

!elapsed_time!

The amount of time that elapsed before the host aircraft and potential threat reach the minimal separation distance assuming no changes in their respective flight characteristics.

!minimal!

The closet distance (range) between the host aircraft and potential threat assuming no changes in their flight characteristics.

Undesired Event Dictionary – none

20. Temporary_Data_Buffers (TDB)

This module provides communication mechanisms between programs for generic message types.

Instantiation Parameters

Parameter	Type	Description
name	identifier	Name for the concrete module.
length	positive	Number of messages of "message_type" the buffer can hold before it is full.
message_type	!message_type!	Message type for the buffer.
consumer	list of !consumer!	List of the names of the consumers and their relative priority.

Instantiation Constraints

1. The !consumer! records for instantiation parameter consumer must be ordered in decreasing probability.

Assumptions

Dependency Assumption

D1. Assignment must be defined on the data type stored in the buffer.

Interface Requirement

I1. This module must permit data to be read from and written into a buffer in a first-in/first-out (FIFO) order.

Concrete Operations

Operation	Parameter	Description	Undesired Events

< if there exists at least one consumer then >

| send | msg:message_type:I
probability:!probability!:I | !in_message! | < forall C in consumer >
  < name >_< C.name >_Overflow
< endfor > |

< forall C in consumer >

| receive_< C.name > | msg:message_type:O | !out_message! | none |

< endfor >

< else >

| send | msg:message_type: I | !in_message! | < name >_Overflow |
| receive | msg:message_type:O | !out_message! | None |

< endif >

Effects

< if there exists at least one consumer then >

send	Adds a message to the FIFO buffer having the specified priority. An exception is raised if the designated priority buffer overflows.

    < forall C in consumer >

receive_< C.name >	Removes the oldest message from the FIFO buffer. The calling program is suspended until a message is available. The service priority of this request is < C.probability >. The request is processed only after all higher priority requests have been processed first.

    < endfor >

< else >

send	Adds a message to the FIFO buffer.
receive	Removes the oldest message from the FIFO buffer. The calling program is suspended until a message is available.

< endif >

Events Signalled – none

Types

message_priority	enumerated (
	     < foreach C in consumer >
	         < C.name >
	     < endfor >
	)

Local Dictionary

!consumer!	record of (
	     name : !cws_id!,
	     priority : integer
	)

	Consumer A has higher probability than consumer B when A.probability > B.probability.
!cws_id!	Enumerated name of the collision warning situation.
!in_message'	The value stored in the buffer.

!message_type!	record of (
	     module : identifier,
	     type : identifier
	)

!out_message!	The value read from the buffer.

Undesired Event Dictionary

< if there exists at least one consumer then >

    < forall C in consumer >

< name >_< C.name >_ Overflow	The named buffer will overflow resulting in loss of data. The message that would cause the overflow is tossed away.

    < endfor >
< else >

< name >_ Overflow	The buffer will overflow resulting in loss of data. The message that would cause the overflow is tossed away.

< endif >

ADAPTABLE DOCUMENTATION COMPONENTS

1. ATD/CWM Software Requirements Specification (SRS)

Instantiation Parameters

Parameter	Type	Description
system	!system_info!	The record contains the project-specific system information.
contract	!contract_info!	The record contains project-specific contract information.
revision	!revision_info!	The record contains document-specific revision information.
alarm	boolean	A **true** value means that the SRS must include engineering requirements describing the audible alarm capability of the ATD/CWM system. A **false** value means that the SRS must omit these requirements.
atc_msg	boolean	A **true** value means that the SRS must include engineering requirements describing the capability of the ATD/CWM system to send an ATC_Msg to the nearest air traffic control center when a collision warning situation has been detected. A **false** value means that the SRS must omit these requirements.
inter_air_msg	boolean	A **true** value means that the SRS must include engineering requirements describing the capability of the ATD/CWM system to send an Inter_Air_Msg to the appropriate potential threat involved in a collision warning situation. A **false** value means that the SRS must omit these requirements.
higher_SRS_spec	identifier	Higher level SRS specification from which the software requirements allocated in this SRS have been derived.

Local Dictionary

```
!contract_info!                          record of (
                                             CDRL_number:   identifier,
                                             agency: identifier,
                                             contract_number:  identifier
                                         )


!revision_info!                          record of (
                                             indicator:  identifier,
                                             date:   identifier
                                         )


!system_info!                            record of (
                                             name:  identifier,
                                             mnemonic: identifier,
                                             id:  identifier
                                         )
```

2. ATD/CWM Interface Requirements Specification (IRS)

Instantiation Parameters

Parameter	Type	Description
system	!system_info!	The record contains the project-specific system information.
contract	!contract_info!	The record contains project-specific contract information.
revision	!revision_info!	The record contains document-specific revision information.
alarm	boolean	A **true** value means that the IRS must include interface requirements describing the role, interface relationships, message formats, and other necessary requirements of the Audible Alarm device interface in the ATD/CWM system. A **false** means that the IRS must omit these requirements.
atc_msg	boolean	A **true** value means that the IRS must include interface requirements describing the role, interface relationships, ATC_Msg message format, and other necessary requirements of the Communication device interface in the ATD/CWM system. A **false** value means that the IRS must omit these requirements.
inter_air_msg	boolean	A **true** value means that the IRS must include interface requirements describing the role, interface relationships, Inter_Air_Msg message format, and other necessary requirements of the Communication device interface in the ATD/CWM system. A **false** means that the IRS must omit these requirements.
mode	enum of (A, C)	A **C** value means that the IRS requirements for the ATC_Msg describe the format of an additional word in the message which contains altitude information. An **A** value means that the IRS must omit these requirements.

Local Dictionary

!contract_info!   record of (
                                            CDRL_number: identifier,
                                            agency: identifier,
                                            contract_number: identifier
)

!revision_info!   record of (
                                            indicator: identifier,
                                            date: identifier
)

!system_info!   record of (
                                            name: identifier,
                                            mnemonic: identifier,
                                            id: identifier
)

3. ATD/CWM Software Design Document (SDD)

Instantiation Parameters

Parameter	Type	Description
system	!system_info!	The record contains the project-specific system information.
contract	!contract_info!	The record contains project-specific contract information.
revision	!revision_info!	The record contains document-specific revision information.
alarm	boolean	A **true** value means that the SDD must include software design information describing how the ATD/CWM system causes the audible alarm to ring (e.g., how, when). A **false** value means the SDD must omit this design information.
atc_msg	boolean	A **true** value means that the SDD must include software design information describing how the ATD/CWM system sends the ATC_Msg to the communication device (e.g., how, when). A **false** value means the SDD must omit this design information.
inter_air_msg	boolean	A **true** value means that the SDD must include software design information describing how the ATD/CWM system sends the Inter_Air_Msg to the communication device (e.g., how, when). A **false** values means the SDD must omit his design information.
temp_buffer	list of !buffer!	Each record in this list describes the name, mnemonic, and hidden decisions of an instance of the Temporary_Data_Buffers module in the ATD/CWM system.

Local Dictionary

!buffer!
```
record of   (
    name : identifier,
    mnemonic : identifier,
    description : text
)
```

!contract_info!
```
record of (
    CDRL_number:    identifier,
    agency: identifier,
    contract_number:  identifier
)
```

!revision_info!   record of (
        indicator: identifier,
        date: identifier
)

!system_info!   record of (
        name: identifier,
        mnemonic: identifier,
        id: identifier
)

Adaptable Verification and Validation Support

Adaptable CSU Test Specifications

1. Audible_Alarm (AA)

Instantiation Parameters

Parameter	Type	Description
ring	list of !ring_info!	Each record defines the pitch and duration at which to ring the audible alarm for a specified collision warning situation.
cws	list of !cws_id!	Names of the collision warning situations.

Instantiation Constraints

1. The duration value for each !ring_info! must have a floating accuracy of exactly two decimal digits. For example. 12.92 is legal: 12.9 and 8 are not legal values.

Local Dictionary

!cws_id!   identifier

!ring_info!   record of (
        cws_name : identifier,
        frequency : integer,
        duration : real
)

2. Collision_Warning_Situation_Status (CWSS)

Instantiation Parameters

Parameter	Type	Description
cws	list of !cws_info!	Each record in this list contains the name and criteria of a collision warning situation.
cws_id	list of identif.er	A list of the names of the collision warning situations specified in the application model.
area	positive	Diameter of the surveillance area
partition	identifier	Name of the concrete module that determines a potential threat partition.

Instantiation Constraints

1. The range value in the !cws_predicate! record must have a floating accuracy of exactly one decimal digit. For example, 12.9 is legal; 12.98 and 8 are not legal.

2. The time value in the !cws_predicate! record must have a floating accuracy of exactly one decimal digit. For example, 30.2 is legal; 30.23 and 30 are not legal.

3. The cws parameter cannot be empty.

4. The records in cws must be ordered in decreasing severity level before instantiating this module.

Local Dictionary

!cws_info!

record of (
    cws_name : identifier,
    severity : real,
    predicate : !cws_predicate!,
    partition : enum of (ID, UID, ALL)
)

!cws_predicate!

union of (
    time : record of (
        time_min : !time_value!,
        time_max : !time_value!),
    range : record of (
        range_min : !range_value!,
        range_max : !range_value!),
    time_and_range : record of (
        (time_min : !time_value!,
        time_max : !time_value!,
        range_min : !range_value!,
        range_max : !range_value!
)
)

!range_value!

Distance from the potential threat to the host aircraft.

!time_value!

How much time elapses before the potential threat and host aircraft reach a separation minima assuming a constant velocity, climb rate, and ground_track.

This page intentionally left blank.

3. GENERATION DESIGN

SOFTWARE GENERATION DESIGN

Decision Model Extensions

This section contains extensions to the Decision Model. An extension is included in this section for one of the following reasons:

- The extension reflects a future variation planned for the Decision Model that is currently defaulted to a value that will remain fixed for this iteration of the Domain Model.

- The extension reflects additional variations on the Adaptable Components that were discovered during the designing and implementing of those components. These extensions may form the basis for Decision Model extensions in future iterations.

Producer_Consumer_Coupling

The Producer_Consumer_Coupling (PCC) describes whether the communication between a message producer and corresponding consumer is tightly- or loosely-coupled. The decision that must be made for this decision class is:

Producer_Consumer_Coupling : and

Loosely-coupled (A **true** value means that the message communication between the producer should be loosely-coupled from the consumer. **False** means tightly-coupled.) : enumerated (true, false)

Message_Buffering

The Message_Buffering (MB) describes what kind of message buffering exists between the message producer and message consumer. It also describes the characteristics of the code component that implements the desired message buffering. The decisions that must be made for this decision class are:

Message_Buffering + : and

Buffer_Name (Name of the buffering code component.) : identifier(1..64)

Mnemonic (Mnemonic for the buffering code component.) : identifier(1..64)

Length (Maximum number of messages that can be stored in the message buffer.) : integer(1..100)

Message_type (Data type of message stored in the buffer.) : and

Module (Name of the module providing the definition of the message data type.) : identifier(1..64)

Type (Data type for the messages stored in the buffer.) : identifier(1..64)

Consumer (Description of the consumers of messages stored in the buffer.)+ : and

Name (Consumer name.) : identifier(1..64)

Priority (Consumer priority. Higher priority is denoted by a target priority value.) : integer

Desc (A textual description what the code component (i.e., the component that implements the desired message buffering) encapsulates and its corresponding hidden decisions.) : text

Minimal_Separation_Distance

The Minimal_Separation_Distance (MSD) describes the minimal separation distance dictated by the FAA. If two inflight aircraft pass each other within this distance, a collision has occurred. The decision that must be made for this decision class is:

Minimal_Separation_Distance : and

distance (Minimal separation distance in feet as dictated by the FAA such that if two aircraft pass each other within this limit, a collision has occurred.) : feet(100..500)

Resolution of Decision Model Extensions

These values must be used exactly as shown for the values for adaptation parameters.

Note: The [x] notation used in the following resolutions differentiates between the multiple instances of the named decision class. The "()" notation designates an empty list.

MSD   (distance: 500.0)

MB[1] (Buffer_Name: Audible_Alarm_Buffer,
          Mnemonic: AAB,
          Length: 10,
          Message_type: (Module: Audible_Alarm_Device,
                            Type: Alarm_Message_Type),
          Consumer: (),
          Desc: "This module encapsulates details about a first-in/first-out buffer to
facilitate loosely-coupled information communication between the audible alarm message producer
and the audible alarm device driver. The hidden decisions of this module are how many entries the
buffer can hold, whether the buffer is of a fixed or varying size, whether the buffer is stored
contiguously in memory or not, and what to do when the buffer is full or empty."
)

MB[2] (Buffer_Name: Communication_Buffer,
          Mnemonic: CB,
          Length: 10,
          Message_type: (Module: Communication_Device,
                            Type: Communication_Msg_Type),
          Consumer: (),
          Desc: "This module encapsulates details about a first-in/first-out buffer to
facilitate loosely-coupled information communication between the communication message producer
and the communication device driver. The hidden decisions of this module are how many entries the
buffer can hold, whether the buffer is of a fixed or varying size, whether the buffer is stored contiguous-
ly in memory or not, and what to do when the buffer is full or empty."
)

MB[3] (Buffer_Name: Radar_Target_Priority_Buffer,
          Mnemonic: RTPB,
          Length: 20,
          Message_type: (Module: Potential_Threat,
                            Type: pt_handle),
          Consumer: (),
          Desc: "This module encapsulates details about a buffering scheme to facilitate
loosely-coupled information communication between a single producer and multiple consumers. The
hidden decisions are how many entries the buffers can hold, whether the buffers are of fixed or varying
size, whether the buffer is stored contiguously in memory or not, and what to do when a buffer is full
or empty."
)

MB[4] (Buffer_Name: Target_Buffer,
          Mnemonic: TB,
          Length: 20,
          Message_type: (Module: Potential_Threat,
                            Type: target_info),
          Consumer: (),
          Desc: "This module encapsulates details about a first-in/first-out buffer to

facilitate loosely-coupled information communication between the radar and ATC devices and the target processor. The hidden decisions of this module are how many entries the buffer can hold, whether the buffer is of a fixed or varying size, whether the buffer is stored contiguously in memory or not, and what to do when the buffer is full or empty."

)

PCC[1]   (Loosely_coupled: True)

PCC[2]   (Loosely_coupled: True)

1. ARCHITECTURE AND COMPONENT MAPPINGS

Table 6-1 collectively presents the Architecture and Component mappings of the software Product Design. The first column of the table names the concrete components that can potentially be included in a generated system. The second column of the table (the Architecture Mapping) describes conditions that must hold for the concrete component to be included in the generated system. References in these conditions (indicated below in **boldface** type) correspond directly to resolutions of either the decision model or the decision model extensions. If a component is to be included in a generated system, then the third column (the Component Mapping) describes which Adaptable Component is to be used to implement the concrete component.

Table 6-1. Software Architecture and Component Mappings

Concrete Component Name	Include this Concrete Component...	Adaptable Component Name
Audible_Alarm_Device	If there is a Collision Warning Situation, C, such that **C.Response.Alarm** is True.	Audible_Alarm_Device
Audible_Alarm_Buffer	If there is (1) a Collision Warning Situation, C, such that **C.Response.Alarm** is True, and (2) **PCC[1].Loosely_coupled** is True.	Temporary_Data_Buffers
Communication_Device	If there is a Collision Warning Situation, C, such that either **C.Response.ATC_Msg** OR **C.Response.Inter_Air_Msg** is True.	Communication_Device
Communication_Buffer	If there is (1) a Collision Warning Situation, C, such that either **C.Response.ATC_Msg** OR **C.Response.Inter_Air_Msg** is True, and (2) **PCC[2].Loosely_coupled** is True.	Temporary_Data_Buffers
Audible_Alarm	If there is a Collision Warning Situation, C, such that **C.Response.Alarm** is True.	Audible_Alarm
Communication	If there is a Collision Warning Situation, C, such that either **C.Response.ATC_Msg** OR **C.Response.Inter_Air_Msg** is True.	Communication
Radar_Target_Priority_Buffer	Always	Temporary_Data_Buffers
Potential_Threat	Always	Potential_Threat

Table 6-1, continued

Concrete Component Name	Include this Concrete Component...	Adaptable Component Name
Target_Buffer	Always	Temporary_Data_Buffers
Host_Aircraft	Always	Host_Aircraft
Initialization_and_Termination	Always	Initialization_and_Termination
Navigation	Always	Navigation
Radar	Always	Radar
Air_Traffic_Control	Always	Air_Traffic_Control
Air_Traffic_Display_Device	Always	Air_Traffic_Display_Device
Collision_Warning_Situation_ Status	Always	Collision_Warning_ Situation_Status
Physical_Quantities	Always	Physical_Quantities
Numerical_Algorithms	Always	Numerical_Algorithms
Air_Traffic_Display	Always	Air_Traffic_Display
Potential_Threat_Partition	If there is a Collision Warning Situation such that **CWS.Partition** is not ALL.	Potential_Threat_Partition
Situation_Dynamics	Always	Situation_Dynamics
Aircraft_Motion	Always	Aircraft_Motion
IHS	Always	IHS
Process_Structure	Always	Process_Structure

2. DECISION MAPPING

Table 6-2 presents the Decision mapping of the software Product Design. The first column of the table names the Concrete Components that can potentially be included in a generated system. The second column of the table lists the adaptation parameters for the Adaptable Component used to implement the Concrete Component (per the Component mapping from Table 6-1). The third column (the Decision Mapping) describes where values for the adaptation parameters are to be obtained. References (indicated below in **boldface** type) correspond directly to resolutions of either the Decision Model or the Decision Model Extensions.

Table 6-2. Software Component Decision Mapping

Concrete Component Name	Parameter		Value is obtained from...
Audible_Alarm_Device	1	Loosely_coupled	**PCC[1].Loosely_coupled**
Audible_Alarm_Buffer	1	Name	**MB[1].Buffer_Name**
	2	Length	**MB[1].Length**
	3	Message_type.Module	**MB[1].Message_type.Module**

Table 6-2, continued

Concrete Component Name		Parameter	Value is obtained from...
Audible_Alarm_Buffer (continued)	4	Message_type.Type	**MB[1].Message_type.Type**
	forall X in **MB[1].Consumer**, aggregate parameters 5 and 6.		
	5	Consumer.Name	**X.Name**
	6	Consumer.Priority	**X.Priority**
Communication_Device	1	Atc_msg	True if there is a Collision Warning Situation, C, such that **C.Response.ATC_Msg** is True. Otherwise, False.
	2	Inter_air_msg	True if there is a Collision Warning Situation, C, such that **C.Response.Inter_Air_Msg** is True. Otherwise, False.
	3	Mode	**ATC_Message.Mode**
	4	Loosely_coupled	**PCC[2].Loosely_coupled**
Communication_Buffer	1	Name	**MB[2].Buffer_Name**
	2	Length	**MB[2].Length**
	3	Message_type. Module	**MB[2].Message_type.Module**
	4	Message_type.Type	**MB[2].Message_type.Type**
	forall X in **MB[2].Consumer**, aggregate parameters 5 and 6.		
	5	Consumer.Name	**X.Name**
	6	Consumer.Priority	**X.Priority**
Audible_Alarm	forall C in CWS such that **C.Response.Alarm** = True, aggregate parameters 1, 2, and 3.		
	1	Ring.cws_name	**C.CWS_Name**
	2	Ring.frequency	**C.Response.Alarm.Pitch**
	3	Ring.duration	**C.Response.Alarm.Duration**
Communication	forall C in CWS such that **C.Response.ATC_Msg** = True, aggregate parameters 1 and 2.		
	1	Atc_msg.cws_name	**C.CWS_Name**
	2	Atc_msg.code	**C.Response.Code**
	forall C in CWS such that **C.Response.Inter_Air_Msg** = True, aggregate parameters 3 and 4.		

Table 6-2, continued

Concrete Component Name		Parameter	Value is obtained from...
Communication (continued)	3	Inter_air_msg. cws_name	**C.CWS_Name**
	4	Inter_air_msg.code	**C.Response.Code**
	5	Mode	**ATC_Message.Mode**
Radar_Target_Priority_Buffer	1	Name	**MB[3].Buffer_Name**
	2	Length	**MB[3].Length**
	3	Message_type.Module	**MB[3].Message_type.Module**
	4	Message_type.Type	**MB[3].Message_type.Type**
	forall C in CWS, aggregate parameters 5 and 6.		
	5	Consumer.Name	**C.CWS_Name**
	6	Consumer.Priority	**C.Severity**
Potential_Threat	forall C in CWS, aggregate parameters 1 through 10.		
	1	Cws_name	**C.CWS_Name**
	2	Severity	**C.Severity**
	Parameters 3.1 and 3.2 are only used when a "time only"-based predicate is used:		
	3.1	Predicate.time.min	**C.CWS_Def.Time.Min**
	3.2	Predicate.time.max	**C.CWS_Def.Time.Max**
	Parameters 4.1 and 4.2 are only used when a "range only"-based predicate is used:		
	4.1	Predicate.range.min	**C.CWS_Def.Range.Min**
	4.2	Predicate.range.max	**C.CWS_Def.Range.Max**
	Parameters 5.1 – 5.4 are used when both a "time and range"-based predicate is used:		
	5.1	Predicate. t_and_r.t_min	**C.CWS_Def.Time.Min**
	5.2	Predicate. t_and_r.t_max	**C.CWS_Def.Time.Max**
	5.3	Predicate. t_and_r.r_min	**C.CWS_Def.Range.Min**
	5.4	Predicate. t_and_r.r_max	**C.CWS_Def.Range.Max**
	6	Partition	**C.Response.Partition**

Table 6-2, continued

Concrete Component Name		Parameter	Value is obtained from...
Potential_Threat (continued)	7	Alarm	**C.Response.Alarm**
	8	Atc_msg	**C.Response.ATC_Msg**
	9	Inter_air_msg	**C.Response.Inter_Air_Msg**
	10	Corrective	**C.Response.Corrective_Msg**
Target_Buffer	1	Name	**MB[4].Buffer_Name**
	2	Length	**MB[4].Length**
	3	Message_type.Module	**MB[4].Message_type.Module**
	4	Message_type.Type	**MB[4].Message_type.Type**
	forall X in **MB[4].Consumer**, aggregate parameters 5 and 6.		
	5	Consumer.Name	**X.Name**
	6	Consumer.Priority	**X.Priority**
Host_Aircraft	None.		
Initialization_and_Termination	None.		
Navigation	None.		
Radar	None.		
Air_Traffic_Control	None.		
Air_Traffic_Display_Device	None.		
Collision_Warning_Situation_Status	forall C in CWS, aggregate parameters 1 through 6.		
	1	CWS_Name	**C.CWS_Name**
	2	Severity	**C.Severity**
	Parameters 3.1 and 3.2 are only used when a "time only"-based predicate is used:		
	3.1	Predicate.time.min	**C.CWS_Def.Time.Min**
	3.2	Predicate.time.max	**C.CWS_Def.Time.Max**
	Parameters 4.1 and 4.2 are only used when a "range only"-based predicate is used:		
	4.1	Predicate.range.min	**C.CWS_Def.Range.Min**
	4.2	Predicate.range.max	**C.CWS_Def.Range.Max**
	Parameters 5.1 - 5.4 are used when both a "time and range"-based predicate is used:		
	5.1	Predicate. t_and_r.t_min	**C.CWS_Def.Time.Min**

Table 6-2, continued

Concrete Component Name	Parameter		Value is obtained from...
Collision_Warning_Situation_ Status (continued)	5.2	Predicate. t_and_r.t_max	**C.CWS_Def.Time.Max**
	5.3	Predicate. t_and_r.r_min	**C.CWS_Def.Range.Min**
	5.4	Predicate. t_and_r.r_max	**C.CWS_Def.Range.Max**
	6	Partition	**C.CWS.Partition**
Physical_Quantities	None.		
Numerical_Algorithms	None.		
Air_Traffic_Display	1	Icon_shape.host_shape	**ADS.Host_Shape**
	2	Icon_shape.id_shape	**ADS.ID_Shape.Shape**
	3	Icon_shape.uid_shape	**ADS.UID_Shape**
	forall X in ASD, aggregate parameters 4 through 11.		
	4	Cws_name	**X.Situation.CWS_Name**
	Parameters 5.1 and 5.2 are only used when a "time only"-based predicate is used:		
	5.1	Predicate.time.min	**X.Situation.CWS_Def.Time.Min**
	5.2	Predicate.time.max	**X.Situation.CWS_Def.Time.Max**
	Parameters 6.1 and 6.2 are only used when a "range only"-based predicate is used:		
	6.1	Predicate.range.min	**X.Situation.CWS_Def.Range.Min**
	6.2	Predicate.range.max	**X.Situation.CWS_Def.Range.Max**
	Parameters 7.1 – 7.4 are used when both a "time and range"-based predicate is used:		
	7.1	Predicate. t_and_r.t_min	**X.Situation.CWS_Def.Time.Min**
	7.2	Predicate. t_and_r.t_max	**X.Situation.CWS_Def.Time.Max**
	7.3	Predicate. t_and_r.r_min	**X.Situation.CWS_Def.Range.Min**
	7.4	Predicate. t_and_r.r_max	**X.Situation.CWS_Def.Range.Max**
	8	Partition	**X.Partition**
	9	Color	**X.PT_Color**
	10	Blink	**X.PT_Blink**

Table 6-2, continued

Concrete Component Name	Parameter		Value is obtained from...
Air_Traffic_Display (continued)	11	Fill	**X.PT_Fill**
	forall X in HASD, aggregate parameters 12 and 13.		
	12	Color.cws_name	**X.Situation.CWS_Name**
	13	Color.color	**X.Color**
	14	Area	**Surveillance_Area.Range**
Potential_Threat_Partition	1	Altitude	True if "altitude" is one of the criteria for identification listed in **ADS.ID_Shape. Partition**. Otherwise, False.
	2	Airspeed	True if "airspeed" is one of the criteria for identification listed in **ADS.ID_Shape. Partition**. Otherwise, False.
Situation_Dynamics	None.		
Aircraft_Motion	1	Msd	**MSD.distance**
IHS	1	Alarm	True if there is a Collision Warning Situation. C, such that **C.Response.Alarm** is True. Otherwise, False.
	2	ATC_Msg	True if there is a Collision Warning Situation. C, such that **C.Response.ATC_Msg** is True. Otherwise, False.
	3	Inter_Air_Msg	True if there is a Collision Warning Situation, C, such that **C.Response.Inter_Air_Msg** is True. Otherwise, False.

Table 6-2, continued

Concrete Component Name	Parameter	Value is obtained from...
IHS (continued)	Construct a list of temp_buffer. There is a temp_buffer for every MB[x] contained in the "Resolutions to the Decision Model" given the following restrictions: • MB[1] used only if **PCC[1].Loosely_coupled** is True and there is a Collision Warning Situation, C, such that **C.Response.Alarm** is True. • MB[2] used only if **PCC[2].Loosely_coupled** is True and there is a Collision Warning Situation, C, such that either **C.Response.ATC_Msg** or **C.Response.Inter_Air_Msg** is True. • MB[3] always used. • MB[4] always used.	
	4.1 Temp_buffer.Name	**MB[x].Buffer_Name**
	4.2 Temp_Buffer. Mnemonic	**MB[x].Mnemonic**
	4.3 Temp_Buffer.Desc	**MB[x].Desc**
Process_Structure	forall C in CWS, aggregate parameters 1 through 5.	
	1 cws.CWS_Name	**C.CWS_Name**
	2 cws.Alarm	**C.Response.Alarm**
	3 cws.ATC_Msg	**C.Response.ATC_Msg**
	4 cws.Inter_Air_Msg	**C.Response.Inter_Air_Msg**
	5 cws.Corrective_Msg	**C.Response.Corrective_Msg**

DOCUMENTATION GENERATION DESIGN

Decision Model Extensions

Revision_Information

The Revision_Information (RI) describes the revision date and document set for all document components of the ATD/CWM domain. The decisions that must be made for this decision class are:

        Revision_Information : and
            date   (Date when the documentation set was generated.) : TBD
            indicator   (Document set indicator.) : identifier(1..64)

Resolution of Decision Model Extensions

TBD

1. ARCHITECTURE AND COMPONENT MAPPINGS

Table 6-3 collectively presents the Architecture and Component mappings of the documentation Product Design. This table has the same organization as Table 6-1.

Table 6-3. Documentation Architecture and Component Mappings

Concrete Document Name	Include this Concrete Component...	Adaptable Document Name
IRS	always	IRS
SRS	always	SRS

2. DECISION MAPPING

Table 6-4 presents the Decision mapping of the documentation Product Design. This table has the same organization as Table 6-2.

Table 6-4. Documentation Component Decision Mapping

Concrete Document Name		Parameter	Value is obtained from...
IRS	1.1	System.Name	**PI.System.Name**
	1.2	System.Mnemonic	**PI.System.Mnemonic**
	1.3	System.Id	**PI.System.ID**
	2.1	Contract.CDRL_Number	**PI.Contract.CDRL**
	2.2	Contract.Agency	**PI.Contract.Agency**
	2.3	Contract.Contract_Number	**PI.Contract.Number**
	3.1	Revision.Indicator	TBD
	3.2	Revision.Date	TBD
	4	Alarm	True if there is a Collision Warning Situation, C, such that **C.Response.Alarm** is True. Otherwise, False.
	5	ATC_Msg	True if there is a Collision Warning Situation, C, such that **C.Response.ATC_Msg** is True. Otherwise, False.
	6	Inter_Air_Msg	True if there is a Collision Warning Situation, C, such that **C.Response.Inter_Air_Msg** is True. Otherwise, False.
	7	Mode	**ATC_Message.Mode**
SRS	1.1	System.Name	**PI.System.Name**

Table 6-4, continued

Concrete Document Name		Parameter	Value is obtained from...
SRS (continued)	1.2	System.Mnemonic	**PI.System.Mnemonic**
	1.3	System.Id	**PI.System.ID**
	2.1	Contract. CDRL_Number	**PI.Contract.CDRL**
	2.2	Contract.Agency	**PI.Contract.Agency**
	2.3	Contract. Contract_Number	**PI.Contract.Number**
	3.1	Revision.Indicator	TBD
	3.2	Revision.Date	TBD
	4	Alarm	True if there is a Collision Warning Situation. C, such that **C.Response.Alarm** is True. Otherwise, False.
	5	ATC_Msg	True if there is a Collision Warning Situation. C, such that **C.Response.ATC_Msg** is True. Otherwise, False.
	6	Inter_Air_Msg	True if there is a Collision Warning Situation, C, such that **C.Response.Inter_Air_Msg** is True. Otherwise, False.

VERIFICATION AND VALIDATION SUPPORT GENERATION DESIGN

Decision Model Extensions

Threat_Partition

The Threat_Partition (TP) describes the name of the concrete code component that determines a potential threat's partition.

> Threat_Partition : and

>> Partition_Module (Name of the concrete module that determines a potential threat's partition.) : identifier(1..64)

Resolution of Decision Model Extensions

TP   (Partition_Module:Potential_Threat)

1. ARCHITECTURE AND COMPONENT MAPPINGS

Table 6-5 collectively presents the Architecture and Component mappings of the verification and validation support Product Design. This table contains the architecture and component mappings for CSU test components. The table has the same organization as Table 6-1.

Table 6-5. Verification and Validation Support Architecture and Component Mappings

Concrete CSU Test Component Name	Include this Concrete Component...	Adaptable CSU Test Component Name
Audible_Alarm	always	AA_CSU
CWSS	always	CWSS_CSU

2. DECISION MAPPING

Table 6-6 presents the Decision mapping of the verification and validation support Product Design. This table has the same organization as Table 6-2.

Table 6-6. Verification and Validation Support Component Decision Mapping

Concrete Component Name	Parameter		Value is obtained from...
Audible_Alarm	forall C in CWS such that **C.Response.Alarm** = True. aggregate parameters 1, 2, and 3.		
	1	Ring.cws_name	**C.CWS_Name**
	2	Ring.frequency	**C.Response.Alarm.Pitch**
	3	Ring.duration	**C.Response.Alarm.Duration**
	forall C in CWS. aggregate parameter 4.		
	4	CWS_Id	**C.CWS_Name**
Collision_Warning_Situation_ Status	forall C in CWS. aggregate parameters 1 through 6.		
	1	CWS_Name	**C.CWS_Name**
	2	Severity	**C.Severity**
	Parameters 3.1 and 3.2 are only used when a "time only"-based predicate is used:		
	3.1	Predicate.time.min	**C.CWS_Def.Time.Min**
	3.2	Predicate.time.max	**C.CWS_Def.Time.Max**
	Parameters 4.1 and 4.2 are only used when a "range only"-based predicate is used:		
	4.1	Predicate.range.min	**C.CWS_Def.Range.Min**
	4.2	Predicate.range.max	**C.CWS_Def.Range.Max**
	Parameters 5.1 - 5.4 are used when both a "time and range"-based predicate is used:		

Table 6-6. continued

Concrete Component Name		Parameter	Value is obtained from...
Collision_Warning_Situation_ Status (continued)	5.1	Predicate. t_and_r.t_min	C.CWS_Def.Time.Min
	5.2	Predicate. t_and_r.t_max	C.CWS_Def.Time.Max
	5.3	Predicate. t_and_r.r_min	C.CWS_Def.Range.Min
	5.4	Predicate. t_and_r.r_max	C.CWS_Def.Range.Max
	6	Partition	C.CWS.Partition
	forall C in CWS, aggregate parameter 7.		
	7	CWS_Id	C.CWS_Name
	8	Partition	TP.Partition_Module
	9	Area	Surveillance_Area.Range

7. ATD/CWM PRODUCT IMPLEMENTATION

1. ADAPTABLE COMPONENTS

ADAPTABLE CODE COMPONENTS

1. Aircraft_Motion (AM)

Spec

```
--
-- Aircraft Motion (AM)
--
-- This module contains programs that model aircraft motion. Aircraft
-- location, velocity, and altitude with respect to the earth and
-- airmass are derived from measures of aircraft motion from devices
-- and other physical modules. The primary hidden decision is the
-- equation of motion.
--
with Physical_Quantities;
generic
    msd : Physical_Quantities.feet;

package Aircraft_Motion is

--
-- Return the FAA dictated minimal separation distance. If two aircraft
-- pass each other within this limit, then a collision has occurred.
--
    function get_msd return Physical_Quantities.feet;


--
-- In three dimensional space, the range between two aircraft
-- can be decomposed into two components: range_xy which is
-- the range component that lies in the X-Y plane; and range_z which
-- is the component lying in the Z plane. This function computes range_xy.
--
    function get_range_xy(distance : in Physical_Quantities.nautical_mile;
                          altitude_A : in Physical_Quantities.feet;
                          altitude_B : in Physical_Quantities.feet)
                                        return
Physical_Quantities.nautical_mile;


--
-- Returns the aircraft's climb_rate (i.e., its vertical velocity).
```

```
-- Exception Climb_Rate_Is_Infinite is raised when time_Y equals time_X.
--

    Climb_Rate_Is_Infinite : exception;

    function get_climb_rate(altitude_Y : in Physical_Quantities.feet;
                            time_Y :     in Physical_Quantities.seconds;
                            altitude_X : in Physical_Quantities.feet;
                            time_X :     in Physical_Quantities.seconds)
                                                       return
Physical_Quantities.fpm;


--
-- In three dimensional space, the velocity of an aircraft can be
-- decomposed into two components: velocity_zy which is the
-- component occurring in the X-Y plane; and velocity_z which
-- is the velocity occurring in the Z plane (i.e., vertical velocity also
-- referred to as climb_rate). This function computes velocity_xy.
--

    function get_velocity_xy(velocity : in Physical_Quantities.knots;
                             rate : in Physical_Quantities.fpm)
                                           return Physical_Quantities.knots;


end Aircraft_Motion;
```

Body

```
--
-- Aircraft_Motion (AM)  package body
--
-- This module contains programs that model aircraft motion. Aircraft
-- location, velocity, and altitude with respect to the earth and
-- airmass are derived from measures of aircraft motion from devices
-- and other physical modules. The primary hidden decision is the
-- equation of motion.
--
with Physical_Quantities; use Physical_Quantities;
with Numerical_Algorithms;
package body Aircraft_Motion is


--
-- Maximum permissible climb_rate value. Used to smooth
-- out gyrations in the get_climb_rate computation.
--

    Max_Climb_Rate : constant Physical_Quantities.fpm := 5000.0;


--
-- Return the FAA dictated minimal separation distance. If two aircraft
-- pass each other within this limit, then a collision has occurred.
--

    function get_msd return Physical_Quantities.feet
    is
    begin
        return msd;
    end get_msd;
```

```
--
-- In three dimensional space, the range between two aircraft
-- can be decomposed into two components: range_xy which is
-- the range component that lies in the X-Y plane; and range_z which
-- is the component lying in the Z plane. This function computes range_xy.
--
    function get_range_xy(distance : in Physical_Quantities.nautical_mile;
                          altitude_A : in Physical_Quantities.feet;
                          altitude_B : in Physical_Quantities.feet)
                                            return
Physical_Quantities.nautical_mile
    is
    begin
        return Physical_Quantities.nautical_mile(
            Numerical_Algorithms.sqrt(distance * distance -
                ((altitude_A -
altitude_B)/Physical_Quantities.nautical_mile_to_feet *
                    (altitude_A -
altitude_B)/Physical_Quantities.nautical_mile_to_feet)));
    end get_range_xy;


--
-- In three dimensional space, the velocity of an aircraft can be
-- decomposed into two components: velocity_zy which is the
-- component occurring in the X-Y plane; and velocity_z which
-- is the velocity occurring in the Z plane (i.e., vertical velocity also
-- referred to as climb_rate). This function computes velocity_xy.
--
    function get_velocity_xy(velocity : in Physical_Quantities.knots;
                             rate : in Physical_Quantities.fpm)
                                            return Physical_Quantities.knots
    is
    begin
        return Physical_Quantities.knots(
          Numerical_Algorithms.sqrt(velocity * velocity -
                    (rate/Physical_Quantities.knot_to_fpm) *
                                (rate/Physical_Quantities.knot_to_fpm)));
    end get_velocity_xy;


--
-- Compute the climb_rate (velocity in the vertical direction) give
-- two altitude readings and the time stamp of each.
-- If the time stamps are equal, then we have a division by zero
-- problem. Thus, we raise exception Climb_Rate_Is_Infinite.
--
    function get_climb_rate(altitude_Y : in Physical_Quantities.feet;
                            time_Y :     in Physical_Quantities.seconds;
                            altitude_X : in Physical_Quantities.feet;
                            time_X :     in Physical_Quantities.seconds)
                                            return
Physical_Quantities.fpm
    is
        climb_rate : float;
```

```
    begin
--
-- By definition, time_Y is always greater than time_X. If
-- time_Y < time_X, then we need to handle a time stamp rollover.
--
        if time_Y < time_X then
            climb_rate := Physical_Quantities.fpm(
                    ((altitude_Y - altitude_X) /
                        float((time_Y - time_X +
Physical_Quantities.seconds'last))) *
                                            Physical_Quantities.fps_to_fpm);
            elsif time_Y = time_X then
                raise Climb_Rate_Is_Infinite;
            else
--
-- altitude/time gives us dimensions of fps (feet per second). So
-- we must convert it to fpm.
--
            climb_rate := Physical_Quantities.fpm(
                        ((altitude_Y - altitude_X) / float((time_Y - time_X))) *
                                        Physical_Quantities.fps_to_fpm);
            end if;
--
-- Adjust rate if necessary.
--
        if climb_rate > Max_Climb_Rate then
            return Max_Climb_Rate;
        else
            return climb_rate;
        end if;
    end get_climb_rate;

end Aircraft_Motion;
```

2. Air_Traffic_Control (ATC)

Spec

```
--
-- Air_Traffic_Control (ATC)  package spec
--
-- This module encapsulates the hardware / software interface
-- to the Air_Traffic_Control device. Its primary hidden decisions
-- are how to obtain raw data for the aircraft_identification, altitude,
-- airspeed, ground track, and range; the scale and format of these
-- input data items; and the device-dependent operations that must be
-- applied to convert the raw data to the internal format of the
-- ATD/CWM system.
--
with Physical_Quantities;
package Air_Traffic_Control is

--
-- Returns information status for a specific aircraft.
```

```
   --
      procedure get_atc_message(aircraft_id : out string,
                                altitude : out Physical_Quantities.feet;
                                airspeed : out Physical_Quantities.knots;
                                ground_track : out Physical_Quantities.degrees;
                                target_range : out
Physical_Quantities.nautical_mile;
                                relative_bearing : out
Physical_Quantities.degrees;
                                timestamp : out Physical_Quantities.seconds);


   end Air_Traffic_Control;
```

Body

```
   --
   -- Air_Traffic_Control (ATC)  package body
   --
   -- This module encapsulates the hardware / software interface
   -- to the Air_Traffic_Control device. Its primary hidden decisions
   -- are how to obtain raw data for the aircraft_identification, altitude,
   -- airspeed, ground track, and range; the scale and format of these
   -- input data items; and the device-dependent operations that must be
   -- applied to convert the raw data to the internal format of the
   -- ATD/CWM system.
   --
   with Physical_Quantities;
   with Simulation_Data;
   package body Air_Traffic_Control is


   --
   -- Returns information status for a specified aircraft.
   --
      procedure get_atc_message(aircraft_id : out string;
                                altitude : out Physical_Quantities.feet;
                                airspeed : out Physical_Quantities.knots;
                                ground_track : out Physical_Quantities.degrees;
                                target_range : out
Physical_Quantities.nautical_mile;
                                relative_bearing : out
Physical_Quantities.degrees;
                                timestamp : out Physical_Quantities.seconds)
      is
      begin
   --
   -- Get information from ATC.
   --
         Simulation_Data.get_sim_data(aircraft_id, altitude, airspeed,
                                      ground_track, target_range,
relative_bearing);
         timestamp := Physical_Quantities.get_time;
      end get_atc_message;
```

```
   end Air_Traffic_Control;
```

3. Air_Traffic_Display_Device (ATDD)

NOTE: The body of this module is implemented Ada and C.

Spec

```
--
-- Air_Traffic_Display_Device (ATDD)  package spec
--
-- This module encapsulates the hardware/software interface to
-- the display. Its primary hidden decisions are the particular sequence
-- of operations necessary to enable and position various icon
-- symbols; the methods for manipulating icon color, shape, shade,
-- and blink characteristics; the method for removing an icon from
-- the display; and the method for writing text to the display.
--
package Air_Traffic_Display_Device is


--
-- Icon shape
--
    type shape is (square, circle, triangle);


--
-- Positioning type
--
    subtype position is integer range -1270 .. 1270;


--
-- Identifier for a created object.
--
    type display_handle is private;
    null_display_handle : constant display_handle;


--
-- Color and Fill type
--
    type colors is (none, red, orange, green, yellow, white, blue,
                    black, pink, purple, indigo, violet);      .

    type fill is new colors;
    type color is new colors;


--
-- Icon size in pixels
--
    subtype size is integer range 1..100;


--
-- Blink type
--
    subtype blink is float digits 1 range 0.0 .. 10.0;
```

```
--
-- Create object. Creates an icon with the given attributes
-- and labels, and returns a handle to it.
--
    function create_object(icon_shape : in shape;
                           icon_size : in size;
                           icon_fill : in fill;
                           icon_color : in color;
                           fill_blink_rate : in blink;
                           obj_blink_rate : in blink;
                           xloc : in position;
                           yloc : in position;
                           label_1 : in string;
                           label_2 : in string;
                           label_3 : in string;
                           label_4 : in string;
                           label_5 : in string) return display_handle;


--
-- Write text to the given location
--
    procedure write_text(msg : in string; xloc : in position; yloc : in
position);


--
-- Set the color of an icon
--
    procedure chg_object_color(id : in display_handle; icon_color : in color);


--
-- Fill an icon
--
    procedure chg_object_fill(id : in display_handle; icon_fill : in fill);


--
-- Blink an icon at the specified rate.
--
    procedure chg_object_blink(id : in display_handle;
                fill_blink_rate : in blink; obj_blink_rate : in blink);


--
-- Set the geometric shape of the icon.
--
    procedure chg_object_shape(id : in display_handle; icon_shape : in shape);


--
-- Move an icon to a new location and update its labels
--
    procedure move_object(id : in display_handle;
                          xloc : in position;
                          yloc : in position;
                          label_1 : in string;
                          label_2 : in string;
                          label_3 : in string;
```

```ada
                                  label_4 : in string;
                                  label_5 : in string);


   --
   -- Delete an object from the display.
   --

      procedure delete_object(id : in display_handle);


   --
   -- Create a display window of a given size at the specified location.
   --
      procedure create_display(xloc : in position; yloc : in position;
                                  width : in position; height : in position);


   private

      type icon_record;
      type display_handle is access icon_record;
      null_display_handle : constant display_handle := null;


   end Air_Traffic_Display_Device;
```

Body (Ada code part)

```ada
   --
   -- Air_Traffic_Display_Device (ATDD)  package body
   --
   -- This module encapsulates the hardware/software interface to
   -- the display. Its primary hidden decisions are the particular sequence
   -- of operations necessary to enable and position various icon
   -- symbols; the methods for manipulating icon color, shape, shade,
   -- and blink characteristics; the method for removing an icon from
   -- the display; and the method for writing text to the display.
   --
   with System;
   with Unchecked_Deallocation;
   with Text_IO;
   package body Air_Traffic_Display_Device is


   --
   -- Label storage and a constant "clear" label.  This will be used to
   -- package the five labels for sending to the C routines.  The string
   -- lengths are currently bounded to 25 characters.
   --

      subtype label is string (1..26);
      clear_label : constant label := label'(1..25 => ' ', others => ASCII.NUL);


   --
   -- Message text storage. Used to store the previous written text message.
   --
      old_msg_text : string(1..35);
      previous_message : boolean := false;
```

```
--
-- Icon information. Record used to store information
-- about an icon displayed on the screen. When an icon
-- is created, a handle is returned to the calling program.
--
-- The icon record stores the following information.
--
--      icon_shape       - icon shape
--      icon_size        - icon size (in pixels)
--      icon_fill        - icon fill color
--      icon_color       - icon border color
--      fill_blink_rate  - how fast the filled interior should blink
--      obj_blink_rate   - how fast the icon itself should blink
--      xloc             - "x" axis location of the upper left corner of the icon
--      yloc             - "y" axis location of the upper left corner of the icon
--      label_1          - First icon label
--      label_2          - Second icon label
--      label_3          - Third icon label
--      label_4          - Fourth icon label
--      label_5          - Fifth icon label
--
    type icon_record is
       record
          icon_shape : shape;
          icon_size : size;
          icon_fill : fill;
          icon_color : color;
          fill_blink_rate : blink;
          obj_blink_rate : blink;
          xloc : position;
          yloc : position;
          label_1 : label;
          label_2 : label;
          label_3 : label;
          label_4 : label;
          label_5 : label;
       end record;


--
-- Unchecked deallocation routine for icon_records.
--
   procedure free is new unchecked_deallocation(icon_record, display_handle);


--
-- Interface declarations to the Xlibrary stuff.
--
   procedure C_Create_Window ( X_Location : in Integer  ;
                               Y_Location : in Integer  ;
                               Width      : in Positive ;
                               Height     : in Positive ) ;

   pragma Interface (C, C_Create_Window) ;
   pragma Import_Procedure (internal => C_Create_Window,
```

```
                                    external => CreateWindow,
                                    parameter_types => (integer,
                                                        integer,
                                                        positive,
                                                        positive),
                                    mechanism => value);

   --
      procedure C_Create_Square ( X_Location : in Integer       ;
                                  Y_Location : in Integer       ;
                                  Side_Size  : in Positive      ;
                                  Fill       : in Natural       ;
                                  Border     : in Natural       ;
                                  Label_1    : in System.Address ;
                                  Label_2    : in System.Address ;
                                  Label_3    : in System.Address ;
                                  Label_4    : in System.Address ;
                                  Label_5    : in System.Address );

      pragma Interface (C, C_Create_Square) ;
      pragma Import_Procedure (internal => C_Create_Square,
                               external => CreateSquare,
                               parameter_types => (integer,
                                                   integer,
                                                   positive,
                                                   natural,
                                                   natural,
                                                   system.address,
                                                   system.address,
                                                   system.address,
                                                   system.address,
                                                   system.address),
                               mechanism => value);

   --
      procedure C_Create_Circle ( X_Location : in Integer       ;
                                  Y_Location : in Integer       :
                                  Diameter   : in Positive      ;
                                  Fill       : in Natural       ;
                                  Border     : in Natural       ;
                                  Label_1    : in System.Address ;
                                  Label_2    : in System.Address ;
                                  Label_3    : in System.Address ;
                                  Label_4    : in System.Address ;
                                  Label_5    : in System.Address );

      pragma Interface (C, C_Create_Circle) ;
      pragma Import_Procedure(internal => C_Create_Circle,
                              external => CreateCircle,
                              parameter_types => (Integer,
                                                  Integer,
                                                  Positive,
                                                  Natural,
                                                  Natural,
```

```
                                             System.Address,
                                             System.Address,
                                             System.Address,
                                             System.Address,
                                             System.Address),
                                 mechanism => value);

--

   procedure C_Create_Triangle ( X_Location : in Integer      ;
                                 Y_Location : in Integer      ;
                                 Height     : in Positive     ;
                                 Fill       : in Natural      ;
                                 Border     : in Natural      ;
                                 Label_1    : in System.Address ;
                                 Label_2    : in System.Address ;
                                 Label_3    : in System.Address ;
                                 Label_4    : in System.Address ;
                                 Label_5    : in System.Address );

   pragma Interface (C, C_Create_Triangle) ;
   pragma Import_Procedure(internal => C_Create_Triangle,
                           external => CreateTriangle,
                           parameter_types => (integer,
                                               integer,
                                               positive,
                                               natural,
                                               natural,
                                               system.address,
                                               system.address,
                                               system.address,
                                               system.address,
                                               system.address),
                           mechanism => value);

--

   procedure C_Draw_Square ( X_Location : in Integer   ;
                             Y_Location : in Integer   ;
                             Side_Size  : in Positive ) ;

   pragma Interface (C, C_Draw_Square) ;
   pragma Import_Procedure( internal => C_Draw_Square,
                            external => DrawSquare,
                            parameter_types => (integer, integer, positive),
                            mechanism => value);

--

   procedure C_Draw_Circle ( X_Location : in Integer   ;
                             Y_Location : in Integer   ;
                             Diameter   : in Positive ) ;

   pragma Interface (C, C_Draw_Circle) ;
   pragma Import_Procedure( internal => C_Draw_Circle,
                            external => DrawCircle,
                            parameter_types => (integer, integer, positive),
                            mechanism => value);
```

```
--

    procedure C_Draw_Triangle ( X_Location : in Integer   ;
                                Y_Location : in Integer   ;
                                Height     : in Positive ) ;

    pragma Interface (C, C_Draw_Triangle) ;
    pragma Import_Procedure(internal => C_Draw_Triangle,
                            external => DrawTriangle,
                            parameter_types => (integer, integer, positive),
                            mechanism => value);


--

    procedure C_Draw_Line ( From_Location_X : in Integer ;
                            From_Location_Y : in Integer ;
                            To_Location_X   : in Integer ;
                            To_Location_Y   : in Integer ) ;

    pragma Interface (C, C_Draw_Line ) ;
    pragma Import_Procedure(internal => C_Draw_Line,
                            external => DrawLine,
                            parameter_types => (integer, integer, integer,
integer),
                            mechanism => value);


--

    procedure C_Draw_String ( X_Location : in Integer      ;
                              Y_Location : in Integer      ;
                              The_String : in System.Address ) ;

    pragma Interface (C, C_Draw_String ) ;
    pragma Import_Procedure(internal => C_Draw_String,
                            external => DrawString,
                            parameter_types => (integer, integer,
system.address),
                            mechanism => value);


--

    procedure C_Move_Triangle ( Height         : in Positive       ;
                                From_X_Location : in Integer        ;
                                From_Y_Location : in Integer        ;
                                To_X_Location   : in Integer        ;
                                To_Y_Location   : in Integer        ;
                                Fill            : in Natural        ;
                                Border          : in Natural        ;
                                Old_Label_1     : in System.Address ;
                                Old_Label_2     : in System.Address ;
                                Old_Label_3     : in System.Address ;
                                Old_Label_4     : in System.Address ;
                                Old_Label_5     : in System.Address ;
                                New_Label_1     : in System.Address ;
                                New_Label_2     : in System.Address ;
                                New_Label_3     : in System.Address ;
                                New_Label_4     : in System.Address ;
                                New_Label_5     : in System.Address );
```

```
    pragma Interface (C, C_Move_Triangle ) ;
    pragma Import_Procedure(internal => C_Move_Triangle,
                            external => MoveTriangle,
                            parameter_types => (positive, integer, integer,
integer,
                                        integer, natural, natural,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address),
                            mechanism => value);


--
    procedure C_Move_Circle ( Diameter         : in Positive       ;
                              From_X_Location : in Integer       ;
                              From_Y_Location : in Integer       ;
                              To_X_Location   . in Integer       ;
                              To_Y_Location   : in Integer       ;
                              Fill            : in Natural       ;
                              Border          : in Natural       ;
                              Old_Label_1     : in System.Address ;
                              Old_Label_2     : in System.Address ;
                              Old_Label_3     : in System.Address ;
                              Old_Label_4     : in System.Address ;
                              Old_Label_5     : in System.Address ;
                              New_Label_1     : in System.Address ;
                              New_Label_2     : in System.Address ;
                              New_Label_3     : in System.Address ;
                              New_Label_4     : in System.Address ;
                              New_Label_5     : in System.Address );


    pragma Interface (C, C_Move_Circle ) ;
    pragma Import_Procedure(internal => C_Move_Circle,
                            external => MoveCircle,
                            parameter_types => (positive, integer, integer,
integer,
                                        integer, natural, natural,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
                                        system.address,
```

```
                                                    system.address),
                                mechanism => value);


    --

    procedure C_Move_Square ( Size              : in Positive        ;
                              From_X_Location : in Integer         ;
                              From_Y_Location : in Integer         ;
                              To_X_Location   : in Integer         ;
                              To_Y_Location   : in Integer         ;
                              Fill              : in Natural         ;
                              Border            : in Natural         ;
                              Old_Label_1       : in System.Address ;
                              Old_Label_2       : in System.Address ;
                              Old_Label_3       : in System.Address ;
                              Old_Label_4       : in System.Address ;
                              Old_Label_5       : in System.Address ;
                              New_Label_1       : in System.Address ;
                              New_Label_2       : in System.Address ;
                              New_Label_3       : in System.Address ;
                              New_Label_4       : in System.Address ;
                              New_Label_5       : in System.Address );

    pragma Interface (C, C_Move_Square ) ;
    pragma Import_Procedure(internal => C_Move_Square,
                            external => MoveSquare,
                            parameter_types => (positive, integer, integer,
integer,
                                                integer, natural, natural,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address,
                                                system.address),
                                mechanism => value);



    --
    -- Local internal routines to simply eliminate the need for duplicate
    -- code. Three routines here: creating a circle, square, or triangle.
    --
    procedure create_circle(handle : in display_handle)
    is
    begin
       c_create_circle(x_location => integer(handle.xloc),
                       y_location => integer(handle.yloc),
                       diameter    => positive(handle.icon_size),
                       fill        => fill'pos(handle.icon_fill),
                       border      => color'pos(handle.icon_color),
```

```
                    label_1    => handle.label_1'address,
                    label_2    => handle.label_2'address,
                    label_3    => handle.label_3'address,
                    label_4    => handle.label_4'address,
                    label_5    => handle.label_5'address);
    end create_circle;

    procedure create_square(handle : in display_handle)
    is
    begin
        c_create_square(x_location => integer(handle.xloc),
                        y_location => integer(handle.yloc),
                        side_size  => positive(handle.icon_size),
                        fill       => fill'pos(handle.icon_fill),
                        border     => color'pos(handle.icon_color),
                        label_1    => handle.label_1'address,
                        label_2    => handle.label_2'address,
                        label_3    => handle.label_3'address,
                        label_4    => handle.label_4'address,
                        label_5    => handle.label_5'address);
    end create_square;

    procedure create_triangle(handle : in display_handle)
    is
    begin
        c_create_triangle(x_location => integer(handle.xloc),
                          y_location => integer(handle.yloc),
                          height     => positive(handle.icon_size),
                          fill       => fill'pos(handle.icon_fill),
                          border     => color'pos(handle.icon_color),
                          label_1    => handle.label_1'address,
                          label_2    => handle.label_2'address,
                          label_3    => handle.label_3'address,
                          label_4    => handle.label_4'address,
                          label_5    => handle.label_5'address);
    end create_triangle;


    --
    -- Create object. Creates an icon with the given attributes
    -- and returns a handle to it.
    --
    function create_object(icon_shape : in shape;
                           icon_size : in size;
                           icon_fill : in fill;
                           icon_color : in color;
                           fill_blink_rate : in blink;
                           obj_blink_rate : in blink;
                           xloc : in position;
                           yloc : in position;
                           label_1 : in string;
                           label_2 : in string;
                           label_3 : in string;
                           label_4 : in string;
```

```
                              label_5 : in string) return display_handle
      is
         handle : display_handle;
      begin
         handle := new icon_record;
         handle.icon_shape := icon_shape;
         handle.icon_size := icon_size;
         handle.icon_fill := icon_fill;
         handle.icon_color := icon_color;
         handle.fill_blink_rate := fill_blink_rate;
         handle.obj_blink_rate := obj_blink_rate;
         handle.xloc := xloc;
         handle.yloc := yloc;
         handle.label_1 := clear_label;
         handle.label_2 := clear_label;
         handle.label_3 := clear_label;
         handle.label_4 := clear_label;
         handle.label_5 := clear_label;
         handle.label_1(1..label_1'length) := label_1;
         handle.label_2(1..label_2'length) := label_2;
         handle.label_3(1..label_3'length) := label_3;
         handle.label_4(1..label_4'length) := label_4;
         handle.label_5(1..label_5'length) := label_5;
         case handle.icon_shape is
            when circle => create_circle(handle);
            when square => create_square(handle);
            when triangle => create_triangle(handle);
         end case;
         return handle;
   exception
      when constraint_error =>
         text_io.put_line("create_object CE");
         return null_display_handle;
      when numeric_error =>
         text_io.put_line("create_object NE");
         return null_display_handle;
      when others =>
         text_io.put_line("create_object Bozo error");
         return null_display_handle;
      end create_object;


   --
   -- Write text to the given location. The previously written message
   -- is erased before writing the new message on the display. Assume that
   -- the xloc and yloc positions of the previous message are exactly
   -- the same as xloc and yloc for the new message.
   --
      procedure write_text(msg : in string; xloc : in position; yloc : in
   position)
      is
      begin
         if previous_message then
            C_Draw_String(x_location => xloc,
```

```
                      y_location => yloc,
                      the_string => old_msg_text'address);
        end if;
        old_msg_text(1..msg'length) := msg;
        old_msg_text(msg'length+1) := Ascii.Nul;
        C_Draw_String(x_location => xloc,
                      y_location => yloc,
                      the_string => old_msg_text'address ) ;
        previous_message := true;
    end write_text;


--
-- Set the color of an icon. Don't do anything if the color
-- is the same.
--
    procedure chg_object_color(id : in display_handle; icon_color : in color)
    is
    begin
        if id.icon_color /= icon_color then
           case id.icon_shape is
              when square =>
                 create_square(id);
                 id.icon_color := icon_color;
                 create_square(id);

              when circle =>
                 create_circle(id);
                 id.icon_color := icon_color;
                 create_circle(id);

              when triangle =>
                 create_triangle(id);
                 id.icon_color := icon_color;
                 create_triangle(id);

           end case;
        end if;
    end chg_object_color;


--
-- Fill an icon. Don't do anything if the fill color is the same.
--
    procedure chg_object_fill(id : in display_handle; icon_fill : in fill)
    is
    begin
        if id.icon_fill /= icon_fill then
           case id.icon_shape is
              when square =>
                 create_square(id);
                 id.icon_fill := icon_fill;
                 create_square(id);

              when circle =>
                 create_circle(id);
```

```
                      id.icon_fill := icon_fill;
                      create_circle(id);

                  when triangle =>
                      create_triangle(id);
                      id.icon_fill := icon_fill;
                      create_triangle(id);

             end case;
         end if;
   end chg_object_fill;


--
-- Blink an icon at the specified rate.
--
   procedure chg_object_blink(id : in display_handle;
                    fill_blink_rate : in blink; obj_blink_rate : in blink)
   is
   begin
      null;
   end chg_object_blink;


--
-- Set the geometric shape of the icon.
--
   procedure chg_object_shape(id : in display_handle; icon_shape : in shape)
   is
   begin
      null;
   end chg_object_shape;


--
-- Move an icon to a new location.
--
   procedure move_object(id : in display_handle;
                          xloc : in position;
                          yloc : in position;
                          label_1 : in string;
                          label_2 : in string;
                          label_3 : in string;
                          label_4 : in string;
                          label_5 : in string)
   is
      temp_label_1, temp_label_2, temp_label_3, temp_label_4, temp_label_5 :
label;
   begin
      temp_label_1 := clear_label,
      temp_label_2 := clear_label;
      temp_label_3 := clear_label;
      temp_label_4 := clear_label;
      temp_label_5 := clear_label;
      temp_label_1(1..label_1'length) := label_1;
      temp_label_2(1..label_2'length) := label_2;
      temp_label_3(1..label_3'length) := label_3;
```

```
        temp_label_4(1..label_4'length) := label_4;
        temp_label_5(1..label_5'length) := label_5;
--
-- See if the location is actually different. If not, then we have
-- nothing to do.
--
        if id.xloc /= xloc or else id.yloc /= yloc
                      or else id.label_1 /= temp_label_1
                      or else id.label_2 /= temp_label_2
                      or else id.label_3 /= temp_label_3
                      or else id.label_4 /= temp_label_4
                      or else id.label_5 /= temp_label_5 then
          case id.icon_shape is
            when circle =>
               C_Move_Circle (Diameter          => positive(id.icon_size),
                              From_X_Location => integer(id.xloc),
                              From_Y_Location => integer(id.yloc),
                              To_X_Location   => integer(xloc),
                              To_Y_Location   => integer(yloc),
                              Fill            => fill'pos(id.icon_fill),
                              Border          => color'pos(id.icon_color),
                              Old_Label_1     => id.label_1'address,
                              Old_Label_2     => id.label_2'address,
                              Old_Label_3     => id.label_3'address,
                              Old_Label_4     => id.label_4'address,
                              Old_Label_5     => id.label_5'address,
                              New_Label_1     => temp_label_1'address,
                              New_Label_2     => temp_label_2'address,
                              New_Label_3     => temp_label_3'address,
                              New_Label_4     => temp_label_4'address,
                              New_Label_5     => temp_label_5'address);

             when square =>
                C_Move_Square (Size            => positive(id.icon_size),
                              From_X_Location => integer(id.xloc),
                              From_Y_Location => integer(id.yloc),
                              To_X_Location   => integer(xloc),
                              To_Y_Location   => integer(yloc),
                              Fill            => fill'pos(id.icon_fill),
                              Border          => color'pos(id.icon_color),
                              Old_Label_1     => id.label_1'address,
                              Old_Label_2     => id.label_2'address,
                              Old_Label_3     => id.label_3'address,
                              Old_Label_4     => id.label_4'address,
                              Old_Label_5     => id.label_5'address,
                              New_Label_1     => temp_label_1'address,
                              New_Label_2     => temp_label_2'address,
                              New_Label_3     => temp_label_3'address,
                              New_Label_4     => temp_label_4'address,
                              New_Label_5     => temp_label_5'address);

              when triangle =>
                 C_Move_Triangle (Height          => positive(id.icon_size),
```

```
                                     From_X_Location => integer(id.xloc),
                                     From_Y_Location => integer(id.yloc),
                                     To_X_Location   => integer(xloc),
                                     To_Y_Location   => integer(yloc),
                                     Fill            => fill'pos(id.icon_fill),
                                     Border          => color'pos(id.icon_color),
                                     Old_Label_1     => id.label_1'address,
                                     Old_Label_2     => id.label_2'address,
                                     Old_Label_3     => id.label_3'address,
                                     Old_Label_4     => id.label_4'address,
                                     Old_Label_5     => id.label_5'address,
                                     New_Label_1     => temp_label_1'address,
                                     New_Label_2     => temp_label_2'address,
                                     New_Label_3     => temp_label_3'address,
                                     New_Label_4     => temp_label_4'address,
                                     New_Label_5     => temp_label_5'address);

         end case;
         id.xloc := xloc;
         id.yloc := yloc;
         id.label_1 := temp_label_1;
         id.label_2 := temp_label_2;
         id.label_3 := temp_label_3;
         id.label_4 := temp_label_4;
         id.label_5 := temp_label_5;
      end if;
exception
   when constraint_error =>
      text_io.put_line("move_object CE");
   when numeric_error =>
      text_io.put_line("move_object NE");
   when others =>
      text_io.put_line("move_object Bozo error");
   end move_object;


--
-- Delete an object from the display.
--
   procedure delete_object(id : in display_handle)
   is
      x : display_handle := id;
   begin
      case x.icon_shape is
         when circle => create_circle(x);
         when square => create_square(x);
         when triangle => create_triangle(x);
      end case;
      free(x);
   end delete_object;


--
-- Create a display window of a given size at the specified location.
--
```

```
      procedure create_display(xloc : in position; yloc : in position;
                               width : in position; height : in position)
      is
      begin
         C_Create_Window(X_Location => integer(xloc),
                         Y_Location => integer(yloc),
                         Width      => positive(width),
                         Height     => positive(height)) ;
      end create_display;

   end Air_Traffic_Display_Device;
```

Body (C code part)

```c
#include <stdio.h>
#include <types.h>
#include <time.h>
#include <stat.h>
#include <signal.h>
#include <X11/Xlib.h>

#ifdef FOOFOO
#define DEBUG(x) fprintf(stderr, x); return;
#else
#define DEBUG(x)
#endif

Display *display;
Window window;
GC grid_gc;
GC icon_gc;
GC text_gc;
Pixmap tile[11], greyscale [12];

void CreateWindow(xspot, yspot, width, height)

  int xspot, yspot;
  unsigned int width, height;

{
  XGCValues grid_gcv;
  XGCValues icon_gcv;

  /* define pixmaps for fill tiles (created using "bitmap" utility) */
  /* tile0 is all white (zeros) */
  static char tile0_bits[] = {
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
  static char tile1_bits[] = {
      0x00, 0x00, 0x80, 0x01, 0xc0, 0x01, 0xe0, 0x01, 0xe0, 0x01, 0x80, 0x01,
      0x80, 0x01, 0x80, 0x01, 0x80, 0x01, 0x80, 0x01, 0x80, 0x01, 0x80, 0x01,
      0x80, 0x01, 0xe0, 0x07, 0xe0, 0x07, 0x00, 0x00};
  static char tile2_bits[] = {
```

```
        0x00, 0x00, 0xe0, 0x0f, 0xf0, 0x1f, 0x38, 0x38, 0x38, 0x30, 0x18, 0x30,
        0x00, 0x38, 0x00, 0x1c, 0x80, 0x0f, 0xe0, 0x07, 0xf0, 0x00, 0x38, 0x00,
        0x1c, 0x00, 0xfc, 0x3f, 0xfc, 0x3f, 0x00, 0x00};
    static char tile3_bits[] = {
        0x00, 0x00, 0xe0, 0x07, 0xf0, 0x0f, 0x70, 0x1e, 0x30, 0x1c, 0x00, 0x1c
        0x00, 0x1c, 0x00, 0x0f, 0x00, 0x0f, 0x00, 0x1c, 0x00, 0x1c, 0x30, 0x1c,
        0x70, 0x1e, 0xf0, 0x0f, 0xe0, 0x07, 0x00, 0x00};
    static char tile4_bits[] = {
        0x00, 0x00, 0x00, 0x0f, 0x80, 0x0f, 0xc0, 0x0d, 0xe0, 0x0c, 0x70, 0x0c,
        0x30, 0x0c, 0x38, 0x0c, 0x18, 0x0c, 0xf8, 0x3f, 0xf8, 0x3f, 0x00, 0x0c,
        0x00, 0x0c, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x00};
    static char tile5_bits[] = {
        0x00, 0x00, 0xf8, 0x1f, 0xf8, 0x1f, 0x18, 0x00, 0x18, 0x00, 0xf8, 0x03,
        0xf8, 0x07, 0x00, 0x0e, 0x00, 0x1c, 0x00, 0x18, 0x00, 0x18, 0x00, 0x1c,
        0x18, 0x0e, 0xf8, 0x07, 0xf0, 0x03, 0x00, 0x00};
    static char tile6_bits[] = {
        0x00, 0x00, 0x00, 0x0c, 0x00, 0x1f, 0xc0, 0x13, 0xe0, 0x00, 0x30, 0x00,
        0x38, 0x00, 0x1c, 0x00, 0x0c, 0x0f, 0xcc, 0x1f, 0xf8, 0x38, 0x78, 0x30,
        0x70, 0x18, 0xe0, 0x1f, 0xc0, 0x07, 0x00, 0x00};
    static char tile7_bits[] = {
        0x00, 0x00, 0xf8, 0x1f, 0xf8, 0x1f, 0x18, 0x1c, 0x00, 0x0e, 0x00, 0x07,
        0x80, 0x03, 0xc0, 0x01, 0xe0, 0x00, 0x60, 0x00, 0x70, 0x00, 0x70, 0x00,
        0x70, 0x00, 0x70, 0x00, 0x70, 0x00, 0x00, 0x00};
    static char tile8_bits[] = {
        0x00, 0x00, 0xc0, 0x03, 0xe0, 0x07, 0x70, 0x0e, 0x38, 0x1c, 0x38, 0x1c,
        0x70, 0x0e, 0xe0, 0x07, 0xe0, 0x07, 0x70, 0x0e, 0x38, 0x1c, 0x38, 0x1c,
        0x70, 0x0e, 0xe0, 0x07, 0xc0, 0x03, 0x00, 0x00};
    static char tile9_bits[] = {
        0x00, 0x00, 0xc0, 0x07, 0xe0, 0x0f, 0x70, 0x0e, 0x38, 0x1c, 0x18, 0x3c,
        0x18, 0x37, 0xf8, 0x33, 0xf0, 0x39, 0x00, 0x38, 0x00, 0x1c, 0x00, 0x1c,
        0x00, 0x0e, 0x00, 0x07, 0x00, 0x07, 0x00, 0x00};
    /* tile10 is all black (ones) */
    static char tile10_bits[] = {
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
    static char grey1_bits[] = {
        0x04, 0x41, 0x00, 0x00, 0x20, 0x08, 0x00, 0x00, 0x82, 0x20, 0x00, 0x00,
        0x08, 0x82, 0x00, 0x00, 0x41, 0x10, 0x00, 0x00, 0x10, 0x04, 0x00, 0x00,
        0x04, 0x41, 0x00, 0x00, 0x20, 0x08, 0x00, 0x00};
    static char grey2_bits[] = {
        0x04, 0x41, 0x41, 0x10, 0x20, 0x08, 0x08, 0x82, 0x82, 0x20, 0x20, 0x08,
        0x08, 0x82, 0x82, 0x20, 0x41, 0x10, 0x08, 0x82, 0x10, 0x04, 0x82, 0x20,
        0x04, 0x41, 0x41, 0x10, 0x20, 0x08, 0x82, 0x20};
    static char grey3_bits[] = {
        0x14, 0x45, 0x41, 0x90, 0x24, 0x09, 0x09, 0x82, 0xc2, 0x24, 0x20, 0x48,
        0x0c, 0x86, 0xa2, 0x20, 0x41, 0x19, 0x88, 0x82, 0x14, 0x24, 0x82, 0x28,
        0x14, 0x45, 0x41, 0x90, 0x28, 0x0a, 0x82, 0x24};
    static char grey4_bits[] = {
        0x54, 0x45, 0x49, 0x92, 0x24, 0x29, 0x19, 0x92, 0xc2, 0x2c, 0x21, 0x49,
        0x4c, 0x86, 0xb2, 0x60, 0x45, 0x19, 0xa8, 0x8a, 0x94, 0x24, 0xa2, 0xa8,
        0x14, 0x4d, 0x45, 0x91, 0x28, 0x4a, 0xa2, 0xa4};
```

```
    static char grey5_bits[] = {
        0x56, 0x4d, 0xc9, 0x92, 0x34, 0x69, 0x19, 0x93, 0xd2, 0xac, 0x25, 0x49,
        0xcc, 0xa6, 0xb2, 0x64, 0x65, 0x59, 0xaa, 0x8a, 0x9c, 0x25, 0xa2, 0xb8,
        0x9c, 0x4d, 0x65, 0x91, 0x28, 0x5b, 0xa6, 0xa4};
    static char grey6_bits[] = {
        0x56, 0x6d, 0xe9, 0x96, 0x36, 0x69, 0x59, 0xb3, 0xda, 0xac, 0xa5, 0x59,
        0xcc, 0xb6, 0xb3, 0x65, 0x6d, 0x59, 0xaa, 0xda, 0xdc, 0x25, 0xb2, 0xba,
        0x9d, 0x4d, 0x6d, 0x95, 0xa8, 0x5b, 0xb6, 0xa6};
    static char grey7_bits[] = {
        0xd6, 0xed, 0xe9, 0x9e, 0x3e, 0x6b, 0xd9, 0xb3, 0xdb, 0xae, 0xb5, 0x59,
        0xce, 0xb7, 0xb3, 0x75, 0x6d, 0xdb, 0xea, 0xda, 0xde, 0xa5, 0xb2, 0xbb,
        0xdd, 0x6d, 0x6d, 0x97, 0xaa, 0x7b, 0xb6, 0xb6};
    static char grey8_bits[] = {
        0xde, 0xed, 0xeb, 0xde, 0xbe, 0x6b, 0xdb, 0xbb, 0xfb, 0xae, 0xb5, 0xdd,
        0xcf, 0xb7, 0xbb, 0x7d, 0x6d, 0xdf, 0xfa, 0xfa, 0xdf, 0xa5, 0xba, 0xfb,
        0xdd, 0x6f, 0x6f, 0xb7, 0xba, 0x7b, 0xb7, 0xb7};
    static char grey9_bits[] = {
        0xdf, 0xef, 0xfb, 0xde, 0xbe, 0xef, 0xdf, 0xbb, 0xfb, 0xef, 0xbd, 0xdd,
        0xef, 0xf7, 0xbb, 0x7f, 0xef, 0xdf, 0xfa, 0xfe, 0xff, 0xad, 0xbb, 0xfb,
        0xfd, 0x7f, 0x6f, 0xf7, 0xfa, 0x7f, 0xbf, 0xb7};
    static char grey10_bits[] = {
        0xdf, 0xff, 0xff, 0xfe, 0xfe, 0xef, 0xdf, 0xff, 0xff, 0xef, 0xfd, 0xfd,
        0xef, 0xff, 0xff, 0x7f, 0xef, 0xff, 0xfb, 0xff, 0xff, 0xbd, 0xbf, 0xff,
        0xfd, 0xff, 0x7f, 0xff, 0xfb, 0x7f, 0xff, 0xdf};

DEBUG("Opened display\n");
    /* open a connection to the X server */
    if ( ( display = XOpenDisplay( 0 ) ) == NULL ) {
      fprintf( stderr, "Cannot connect to X server %s.\n", getenv( "DISPLAY" )
);

      exit( 1 );
    }

    window = XCreateSimpleWindow(
              display, DefaultRootWindow( display ),
              xspot, yspot, width, height, 3,
              BlackPixel( display, DefaultScreen( display ) ),
              WhitePixel( display, DefaultScreen( display ) ) );

    XSelectInput( display, window,  ExposureMask );

    XMapWindow( display, window );

    XFlush( display );

    grid_gcv.foreground = BlackPixel( display, DefaultScreen( display ) );
    grid_gcv.background = WhitePixel( display, DefaultScreen( display ) );

    grid_gc = XCreateGC( display, window, GCForeground | GCBackground, &grid_gcv
);

    /* see p-61,62 for definition of the XGCValues data type */
    icon_gcv.foreground = BlackPixel( display, DefaultScreen( display ) );
```

```
   icon_gcv.background = WhitePixel( display, DefaultScreen( display ) );
   icon_gcv.function = GXxor;
   icon_gcv.graphics_exposures = False;

   icon_gc = XCreateGC( display, window,
                        GCForeground | GCBackground | GCFunction |
GCGraphicsExposures,
                        &icon_gcv );

   /* text_gc uses same initial values as icon_gc */
   text_gc = XCreateGC( display, window,
                        GCForeground | GCBackground | GCFunction |
GCGraphicsExposures,
                        &icon_gcv );

   XClearWindow( display, window );

   tile[0] = XCreateBitmapFromData( display, window, tile0_bits, 16, 16 );
   tile[1] = XCreateBitmapFromData( display, window, tile1_bits, 16, 16 );
   tile[2] = XCreateBitmapFromData( display, window, tile2_bits, 16, 16 );
   tile[3] = XCreateBitmapFromData( display, window, tile3_bits, 16, 16 );
   tile[4] = XCreateBitmapFromData( display, window, tile4_bits, 16, 16 );
   tile[5] = XCreateBitmapFromData( display, window, tile5_bits, 16, 16 );
   tile[6] = XCreateBitmapFromData( display, window, tile6_bits, 16, 16 );
   tile[7] = XCreateBitmapFromData( display, window, tile7_bits, 16, 16 );
   tile[8] = XCreateBitmapFromData( display, window, tile8_bits, 16, 16 );
   tile[9] = XCreateBitmapFromData( display, window, tile9_bits, 16, 16 );
   tile[10] = XCreateBitmapFromData( display, window, tile10_bits, 16, 16 );

   greyscale[0]  = XCreateBitmapFromData( display, window, tile0_bits, 16, 16
);
   greyscale[1]  = XCreateBitmapFromData( display, window, grey1_bits, 16, 16
);
   greyscale[2]  = XCreateBitmapFromData( display, window, grey2_bits, 16, 16
);
   greyscale[3]  = XCreateBitmapFromData( display, window, grey3_bits, 16, 16
);
   greyscale[4]  = XCreateBitmapFromData( display, window, grey4_bits, 16, 16
);
   greyscale[5]  = XCreateBitmapFromData( display, window, grey5_bits, 16, 16
);
   greyscale[6]  = XCreateBitmapFromData( display, window, grey6_bits, 16, 16
);
   greyscale[7]  = XCreateBitmapFromData( display, window, grey7_bits, 16, 16
);
   greyscale[8]  = XCreateBitmapFromData( display, window, grey8_bits, 16, 16
);
   greyscale[9]  = XCreateBitmapFromData( display, window, grey9_bits, 16, 16
);
   greyscale[10] = XCreateBitmapFromData( display, window, grey10_bits, 16, 16
);
   greyscale[11] = XCreateBitmapFromData( display, window, tile10_bits, 16, 16
);
```

```
    XFlush( display );
}

/* -------------------------------------------------- */

 void CreateCircle (xspot, yspot, diameter, fill, border, s1, s2, s3, s4, s5)

   int xspot, yspot;
   unsigned int diameter;
   int fill, border;
   char *s1, *s2, *s3, *s4, *s5;

{
   extern Display *display;
   extern Window window;
DEBUG("Created circle\n");
   XSetFillStyle( display, icon_gc, FillTiled );
   XSetTile( display, icon_gc, greyscale[fill] );
   XFillArc ( display, window, icon_gc, xspot, yspot, diameter, diameter, 0,
23040 ) ;

   XSetLineAttributes ( display, icon_gc, 2, border, CapRound, JoinRound ) ;
   XSetTile( display, icon_gc, greyscale[11] );
   XDrawArc ( display, window, icon_gc, xspot, yspot, diameter, diameter, 0,
23040 ) ;

   XDrawString( display, window, text_gc, xspot+diameter+10, yspot, s1,
strlen(s1) );
   XDrawString( display, window, text_gc, xspot+diameter+10, yspot+12, s2,
strlen(s2) );
   XDrawString( display, window, text_gc, xspot+diameter+10, yspot+24, s3,
strlen(s3) );
   XDrawString( display, window, text_gc, xspot+diameter+10, yspot+36, s4,
strlen(s4) );
   XDrawString( display, window, text_gc, xspot+diameter+10, yspot+48, s5,
strlen(s5) );

   XFlush( display );   /* causes requests to be processed by X server */
}

/* -------------------------------------------------- */

void CreateSquare (xspot, yspot, size, fill, border, s1, s2, s3, s4, s5)

   int xspot, yspot;
   unsigned int size;
   int fill, border;
   char *s1, *s2, *s3, *s4, *s5;

{
   extern Display *display;
   extern Window window;
DEBUG("Created square\n");
   XSetFillStyle( display, icon_gc, FillTiled );
```

```
    XSetTile( display, icon_gc, greyscale[fill] );
    XFillRectangle( display, window, icon_gc, xspot, yspot, size, size ) ;

    XSetLineAttributes ( display, icon_gc, 2, border, CapRound, JoinRound ) ;
    XSetTile( display, icon_gc, greyscale[11] );
    XDrawRectangle( display, window, icon_gc, xspot, yspot, size, size ) ;

    XDrawString( display, window, text_gc, xspot+size+10, yspot, s1, strlen(s1)
);
    XDrawString( display, window, text_gc, xspot+size+10, yspot+12, s2,
strlen(s2) );
    XDrawString( display, window, text_gc, xspot+size+10, yspot+24, s3,
strlen(s3) );
    XDrawString( display, window, text_gc, xspot+size+10, yspot+36, s4,
strlen(s4) );
    XDrawString( display, window, text_gc, xspot+size+10, yspot+48, s5,
strlen(s5) );

    XFlush( display );  /* causes requests to be processed by X server */
}

/* ---------------------------------------------------- */

void CreateTriangle (xspot, yspot, height, fill, border, s1, s2, s3, s4, s5)

    int xspot, yspot;
    unsigned int height;
    int fill, border;
    char *s1, *s2, *s3, *s4, *s5;

{
    XPoint points[4];

    extern Display *display;
    extern Window window;
DEBUG("Created triangle\n");
    XSetFillStyle( display, icon_gc, FillTiled );
    XSetTile( display, icon_gc, greyscale[fill] );

    /* points[0] = upper tip of triangle */
    points[0].x = xspot + (height/2);
    points[0].y = yspot;

    /* points[1] = lower left corner of triangle */
    points[1].x = xspot;
    points[1].y = yspot + height;

    /* points[2] = lower right corner of triangle */
    points[2].x = xspot + height;
    points[2].y = yspot + height;

    /* points[3] = upper tip of triangle */
    points[3].x = xspot + (height/2);
    points[3].y = yspot;
```

```
  XFillPolygon ( display, window, icon_gc, points, 4, Convex, CoordModeOrigin
) ;

  XSetLineAttributes ( display, icon_gc, 2, border, CapRound, JoinRound ) ;
  XSetTile( display, icon_gc, greyscale[11] );
  XDrawLine( display, window, icon_gc, points[0].x, points[0].y, points[1].x,
points[1].y);
  XDrawLine( display, window, icon_gc, points[1].x, points[1].y, points[2].x,
points[2].y);
  XDrawLine( display, window, icon_gc, points[2].x, points[2].y, points[0].x,
points[0].y);

  XDrawString( display, window, text_gc, xspot+height+10, yspot, s1,
strlen(s1) );
  XDrawString( display, window, text_gc, xspot+height+10, yspot+12, s2,
strlen(s2) );
  XDrawString( display, window, text_gc, xspot+height+10, yspot+24, s3,
strlen(s3) );
  XDrawString( display, window, text_gc, xspot+height+10, yspot+36, s4,
strlen(s4) );
  XDrawString( display, window, text_gc, xspot+height+10, yspot+48, s5,
strlen(s5) );

  XFlush( display );  /* causes requests to be processed by X server */
}

/* -------------------------------------------------- */

void DrawLine (xspot1, yspot1, xspot2, yspot2)

  int xspot1, yspot1, xspot2, yspot2;

{
  extern Display *display;
  extern Window window;
DEBUG("Draw line\n");
  XDrawLine( display, window, grid_gc, xspot1, yspot1, xspot2, yspot2 );

  XFlush( display );  /* causes requests to be processed by X server */
}

/* -------------------------------------------------- */

void DrawCircle (xspot, yspot, diameter)

  int xspot, yspot;
  unsigned int diameter;

{
  extern Display *display;
  extern Window window;
DEBUG("Draw circle\n");

  XDrawArc( display, window, grid_gc, xspot, yspot, diameter, diameter, 0,
23040 );
```

```
  XFlush( display );   /* causes requests to be processed by X server */
}

/* ---------------------------------------------------- */

void DrawSquare (xspot, yspot, size)

  int xspot, yspot;
  unsigned int size;

{
  extern Display *display;
  extern Window window;
DEBUG("Draw square\n");

  XDrawRectangle( display, window, grid_gc, xspot, yspot, size, size);

  XFlush( display );   /* causes requests to be processed by X server */
}

/* ---------------------------------------------------- */

void DrawTriangle (xspot, yspot, height)

  int xspot, yspot;
  unsigned int height;

{
  extern Display *display:
  extern Window window;
DEBUG("Draw triangle\n");

  /* draw line from top to bottom left */
  XDrawLine( display, window, grid_gc, xspot+(height/2), yspot, xspot,
yspot+height);

  /* draw line from bottom left to bottom right */
  XDrawLine( display, window, grid_gc, xspot, yspot+height, xspot+height,
yspot+height);

  /* draw line from bottom right to top */
  XDrawLine( display, window, grid_gc, xspot+height, yspot+height,
xspot+(height/2), yspot);

  XFlush( display );   /* causes requests to be processed by X server */
}

/* ---------------------------------------------------- */

void DrawString (xspot, yspot, s)

  int xspot, yspot:
  char *s;

{
  extern Display *display;
```

```
    extern Window window;
DEBUG("Draw string\n");

  XDrawString( display, window, text_gc, xspot, yspot, s, strlen(s) );

  XFlush( display );  /* causes requests to be processed by X server */
}

/* --------------------------------------------------- */

void MoveSquare (size, fromxspot, fromyspot, toxspot, toyspot, fill, border,
olds1, olds2, olds3, olds4, olds5, s1, s2, s3, s4, s5)

  int size, fromxspot, fromyspot, toxspot, toyspot, fill, border;
  char *olds1, *olds2, *olds3, *olds4, *olds5, *s1, *s2, *s3, *s4, *s5;

{
  extern Display *display;
  extern Window window;
DEBUG("Move square\n");
  XSetFillStyle( display, icon_gc, FillTiled );
  XSetLineAttributes ( display, icon_gc, 2, border, CapRound, JoinRound ) ;

  /* erase old square */
  XSetTile( display, icon_gc, greyscale[fill] );
  XFillRectangle( display, window, icon_gc, fromxspot, fromyspot, size, size )
;

  XSetTile( display, icon_gc, greyscale[11] );
  XDrawRectangle( display, window, icon_gc, fromxspot, fromyspot, size, size )
;

  /* erase old label */
  XDrawString( display, window, text_gc, fromxspot+size+10, fromyspot, olds1,
strlen(olds1) );
  XDrawString( display, window, text_gc, fromxspot+size+10, fromyspot+12,
olds2, strlen(olds2) );
  XDrawString( display, window, text_gc, fromxspot+size+10, fromyspot+24,
olds3, strlen(olds3) );
  XDrawString( display, window, text_gc, fromxspot+size+10, fromyspot+36,
olds4, strlen(olds4) );
  XDrawString( display, window, text_gc, fromxspot+size+10, fromyspot+48,
olds5, strlen(olds5) );

  /* draw new square */
  XSetTile( display, icon_gc, greyscale[fill] );
  XFillRectangle( display, window, icon_gc, toxspot, toyspot, size, size ) ;

  XSetTile( display, icon_gc, greyscale[11] );
  XDrawRectangle( display, window, icon_gc, toxspot, toyspot, size, size ) ;

  XDrawString( display, window, text_gc, toxspot+size+10, toyspot, s1,
strlen(s1) );
  XDrawString( display, window, text_gc, toxspot+size+10, toyspot+12, s2,
strlen(s2) );
```

```c
  XDrawString( display, window, text_gc, toxspot+size+10, toyspot+24, s3,
strlen(s3) );
  XDrawString( display, window, text_gc, toxspot+size+10, toyspot+36, s4,
strlen(s4) );
  XDrawString( display, window, text_gc, toxspot+size+10, toyspot+48, s5,
strlen(s5) );

  XFlush( display );
}

/* ------------------------------------------------- */

void MoveCircle ( diameter, fromxspot, fromyspot, toxspot, toyspot, fill,
border, olds1, olds2, olds3, olds4, olds5, s1, s2, s3, s4, s5)

  int diameter, fromxspot, fromyspot, toxspot, toyspot, fill, border;
  char *olds1, *olds2, *olds3, *olds4, *olds5, *s1, *s2, *s3, *s4, *s5;

{
  extern Display *display;
  extern Window window;
DEBUG("Move circle\n");

  XSetFillStyle( display, icon_gc, FillTiled );
  XSetLineAttributes ( display, icon_gc, 2, border, CapRound, JoinRound ) ;

  /* erase old circle */
  XSetTile( display, icon_gc, greyscale[fill] );
  XFillArc ( display, window, icon_gc, fromxspot, fromyspot, diameter,
diameter, 0, 23040 ) ;

  XSetTile( display, icon_gc, greyscale[11] );
  XDrawArc ( display, window, icon_gc, fromxspot, fromyspot, diameter,
diameter, 0, 23040 ) ;

  /* erase old label */
  XDrawString( display, window, text_gc, fromxspot+diameter+10, fromyspot,
olds1, strlen(olds1) );
  XDrawString( display, window, text_gc, fromxspot+diameter+10, fromyspot+12,
olds2, strlen(olds2) );
  XDrawString( display, window, text_gc, fromxspot+diameter+10, fromyspot+24,
olds3, strlen(olds3) );
  XDrawString( display, window, text_gc, fromxspot+diameter+10, fromyspot+36,
olds4, strlen(olds4) );
  XDrawString( display, window, text_gc, fromxspot+diameter+10, fromyspot+48,
olds5, strlen(olds5) );

  /* draw new circle */
  XSetTile( display, icon_gc, greyscale[fill] );
  XFillArc ( display, window, icon_gc, toxspot, toyspot, diameter, diameter,
0, 23040 ) ;

  XSetTile( display, icon_gc, greyscale[11] );
  XDrawArc ( display, window, icon_gc, toxspot, toyspot, diameter, diameter,
0, 23040 ) ;
```

```
  /* draw new label */
  XDrawString( display, window, text_gc, toxspot+diameter+10, toyspot, s1,
strlen(s1) );
  XDrawString( display, window, text_gc, toxspot+diameter+10, toyspot+12, s2,
strlen(s2) );
  XDrawString( display, window, text_gc, toxspot+diameter+10, toyspot+24, s3,
strlen(s3) );
  XDrawString( display, window, text_gc, toxspot+diameter+10, toyspot+36, s4,
strlen(s4) );
  XDrawString( display, window, text_gc, toxspot+diameter+10, toyspot+48, s5,
strlen(s5) );

  XFlush( display );
}

/* ----------------------------------------------------- */

void MoveTriangle (height, fromxspot, fromyspot, toxspot, toyspot, fill,
border, olds1, olds2, olds3, olds4, olds5, s1, s2, s3, s4, s5)

  int height, fromxspot, fromyspot, toxspot, toyspot, fill, border;
  char *olds1, *olds2, *olds3, *olds4, *olds5, *s1, *s2, *s3, *s4, *s5;

{
  extern Display *display;
  extern Window window;

  XPoint points[4];
DEBUG("Move triangle\n");

  /* points[0] = upper tip of triangle */
  points[0].x = fromxspot + (height/2);
  points[0].y = fromyspot;

  /* points[1] = lower left corner of triangle */
  points[1].x = fromxspot;
  points[1].y = fromyspot + height;

  /* points[2] = lower right corner of triangle */
  points[2].x = fromxspot + height;
  points[2].y = fromyspot + height;

  /* points[3] = upper tip of triangle */
  points[3].x = fromxspot + (height/2);
  points[3].y = fromyspot;

  /* erase old icon */
  XSetLineAttributes ( display, icon_gc, 2, border, CapRound, JoinRound ) ;
  XSetFillStyle( display, icon_gc, FillTiled );

  XSetTile( display, icon_gc, greyscale[fill] );
  XFillPolygon ( display, window, icon_gc, points, 4, Convex, CoordModeOrigin
) ;

  XSetTile( display, icon_gc, greyscale[11] );
  XDrawLine( display, window, icon_gc, points[0].x,  points[0].y, points[1].x,
```

```
points[1].y ) ;
  XDrawLine( display, window, icon_gc, points[1].x,  points[1].y, points[2].x,
points[2].y ) ;
  XDrawLine( display, window, icon_gc, points[2].x,  points[2].y, points[3].x,
points[3].y ) ;

  XDrawString( display, window, text_gc, fromxspot+height+10, fromyspot,
olds1, strlen(olds1) );
  XDrawString( display, window, text_gc, fromxspot+height+10, fromyspot+12,
olds2, strlen(olds2) );
  XDrawString( display, window, text_gc, fromxspot+height+10, fromyspot+24,
olds3, strlen(olds3) );
  XDrawString( display, window, text_gc, fromxspot+height+10, fromyspot+36,
olds4, strlen(olds4) );
  XDrawString( display, window, text_gc, fromxspot+height+10, fromyspot+48,
olds5, strlen(olds5) );

  /* points[0] = upper tip of triangle */
  points[0].x = toxspot + (height/2);
  points[0].y = toyspot;

  /* points[1] = lower left corner of triangle */
  points[1].x = toxspot;
  points[1].y = toyspot + height;

  /* points[2] = lower right corner of triangle */
  points[2].x = toxspot + height;
  points[2].y = toyspot + height;

  /* points[3] = upper tip of triangle */
  points[3].x = toxspot + (height/2);
  points[3].y = toyspot;

  /* draw new icon */
  XSetTile( display, icon_gc, greyscale[fill] );
  XFillPolygon ( display, window, icon_gc, points, 4, Convex, CoordModeOrigin
) ;

  XSetTile( display, icon_gc, greyscale[11] );
  XDrawLine( display, window, icon_gc, points[0].x,  points[0].y, points[1].x,
points[1].y ) ;
  XDrawLine( display, window, icon_gc, points[1].x,  points[1].y, points[2].x,
points[2].y ) ;
  XDrawLine( display, window, icon_gc, points[2].x,  points[2].y, points[3].x,
points[3].y ) ;

  XDrawString( display, window, text_gc, toxspot+height+10, toyspot, s1,
strlen(s1) );
  XDrawString( display, window, text_gc, toxspot+height+10, toyspot+12, s2,
strlen(s2) );
  XDrawString( display, window, text_gc, toxspot+height+10, toyspot+24, s3,
strlen(s3) );
  XDrawString( display, window, text_gc, toxspot+height+10, toyspot+36, s4,
strlen(s4) );
```

```
   XDrawString( display, window, text_gc, toxspot+height+10, toyspot+48, s5,
strlen(s5) );

   XFlush( display );
}


/* ----------------------------------------------------- */
/*
void SetDisplayType (iscolor)

   int iscolor;

{
   extern Display *display;
   extern Window window;

   XSetWindowColormap ( display, window,
                        DefaultColormap ( display, window ) ) ;

   XFlush( display );
} */
```

4. Air_Traffic_Display (ATD)

Spec

```
   --
   -- Air_Traffic_Display (ATD) package spec
   --
   -- This module knows how to determine the content of the
   -- corrective action advisory message, what aircraft status
   -- information to display, and where to position aircraft
   -- symbols on the Air_Traffic_Display device.
   --
   with Potential_Threat;
   package Air_Traffic_Display is


   --
   -- Determine the content of the corrective action advisory message
   -- (Corrective_Msg) and have it shown on the air traffic display.
   --
      procedure corrective_action_msg(threat : in Potential_Threat.pt_handle);


   --
   -- Update an aircraft's display symbol when the aircraft partition
   -- changes.
   --
      procedure update_ads(threat : in Potential_Threat.pt_nandle);


   --
   -- Initialize the air traffic display
   --
      procedure initialize_display;
```

```
--
-- Update the displayed information for the specified potential threat.
--
    procedure update_cws(threat : in Potential_Threat.pt_handle;
                          display_status : in Potential_Threat.target_display);

end Air_Traffic_Display;
```

Body

```
--
-- Air_Traffic_Display (ATD) package body
--
-- This module knows how to determine the content of the
-- corrective action advisory message, what aircraft status
-- information to display, and where to position aircraft
-- symbols on the Air_Traffic_Display device.
--
with Air_Traffic_Display_Device; use Air_Traffic_Display_Device;
with Potential_Threat; use Potential_Threat;
with Host_Aircraft;
with Situation_Dynamics;
with Air_Craft_Motion;
with Physical_Quantities;
with Numerical_Algorithms;
with Text_IO;
package body Air_Traffic_Display is

    package float_io is new text_io.float_io(float); use float_io;


--
-- Screen dimensions given in the following terms:
--
--   x_origin - "X" axis location of the display.
--   y_origin - "Y" axis location of the display.
--   x_size   - Horizontal length across the screen of the display.
--   y_size   - Vertical length down the screen of the display.
--   x_center - "X" axis location of the center of the screen.
--   y_center - "Y" axis location of the center of the screen.
--
--   (0, 0) is the upper left hand corner.
--
-- Start location of corrective action advisory message in the display.
--
--   x_msg    - "X" axis location
--   y_msg    - "Y" axis location
--
    x_origin : constant Air_Traffic_Display_Device.position := 0;
    y_origin : constant Air_Traffic_Display_Device.position := 0;
    x_size   : constant Air_Traffic_Display_Device.position := 1270;
    y_size   : constant Air_Traffic_Display_Device.position := 1000;
    x_center : constant Air_Traffic_Display_Device.position := 635;    --
x_size / 2;
    y_center : constant Air_Traffic_Display_Device.position := 500;    --
```

```
y_size / 2;
   x_msg : constant Air_Traffic_Display_Device.position := 610;        --
x_center - 25;
   y_msg : constant Air_Traffic_Display_Device.position := 980;        --
y_size - 20;
   icon_size : constant Air_Traffic_Display_Device.size := 16;

--
-- Host aircraft display handle.
--
   host_aircraft_handle : Air_Traffic_Display_Device.display_handle;

--
-- Determine the contents of the Corrective_Action_Msg and send
-- it to the Air_Traffic_Display_Device (ATDD).
--
   msg_1 : constant string := "Maintain current heading and rate";
   msg_2 : constant string := "Fly level";
   msg_3 : constant string := "Climb at ";
   msg_4 : constant string := "Dive at ";
   msg_suffix : constant string := " ft/min";

   subtype message_index is integer range 0..8;

   package Nautical_Mile_IO is new Text_IO.Float_IO(num =>
Physical_Quantities.nautical_mile);

--
-- Local formatting routine for the corrective action advisory message.
--
   procedure concatenate(text1 : in string;
                          text2 : in string; value : in
Physical_Quantities.fpm)
   is
   begin
      Air_Traffic_Display_Device.write_text(text1 &
                   integer'image(integer(value)) & text2, x_msg, y_msg);
   end concatenate;

--
-- Local formatting routine for outputting the range label (in floating pt.
format).
--
   function nautical_mile_to_string(value : in
Physical_Quantities.nautical_mile) return string
   is
      temp_string : string (1..10);
      out_string  : string(1..11);
   begin
      Nautical_Mile_IO.Put(To    => temp_string,
                           Item => value, aft => 1, exp => 0);
      out_string(1..10) := temp_string;
      out_string(11) := ASCII.NUL;
      return (out_string);
   end nautical_mile_to_string;
```

```
--
-- Function to determine whether a quantity is less than zero, equal to zero,
-- or greater than zero. The following values are returned.
--
--    = 0 => 0
--    < 0 => 1
--    > 0 => 2
--
    function zero_test(value : in Physical_Quantities.fpm) return integer
    is
    begin
       if value = 0.0 then
          return 0;
       elsif value < 0.0 then
          return 1;
       else
          return 2;
       end if;
    end zero_test;


--
-- Format a message informing the pilot of the host_aircraft
-- how to avoid a collision warning situation.
--
    procedure corrective_action_msg(threat : in Potential_Threat.pt_handle)
    is
       msd : Physical_Quantities.feet;          -- minimal separation distance
       FAA_msd : Physical_Quantities.feet;      -- FAA approved minimal
separation distance
       pt_rate : Physical_Quantities.fpm;       -- climb rate for
potential_threat
       pt_altitude : Physical_Quantities.feet;-- altitude for potential_threat
       ha_rate : Physical_Quantities.fpm;       -- climb rate for host_aircraft
       ha_altitude : Physical_Quantities.feet;-- altitude for host_aircraft
       change_rate : Physical_Quantities.fpm; -- climb/descend rate
    begin
begin
       msd := Situation_Dynamics.get_msd(threat);
exception
   when numeric_error => text_io.put_line("cam 0 - NE");
   when constraint_error => text_io.put_line("cam 0 - CE");
   when others => text_io.put_line("cam 0 - Bozo error");
end;
       FAA_msd := Air_Craft_Motion.get_msd;
--
-- If the predicted minimal separation distance is no less than the
-- minimal distance dictated by the FAA, then no change in heading
-- or climb rate is necessary.
--
       if msd >= FAA_msd then
          Air_Traffic_Display_Device.write_text(msg_1, x_msg, y_msg);
          return;
       end if;
```

```
--
-- If minimal separation distance will be less than the FAA dictated
-- distance, then the appropriate corrective advisory message is
-- a function of the climb_rate and altitude of the
-- potential threat and host aircraft.
--
begin
      ha_altitude := Host_Aircraft.get_altitude;
      ha_rate := Host_Aircraft.get_climb_rate;
      pt_altitude := Potential_Threat.get_altitude(threat);
      pt_rate := Potential_Threat.get_climb_rate(threat);
exception
   when numeric_error => text_io.put_line("cam 1 - NE");
   when constraint_error => text_io.put_line("cam 1 - CE");
   when others => text_io.put_line("cam 1 - Bozo error");
end;


--
-- Determine what message to send based on the current altitudes
-- and climb_rates of the potential threat and host aircraft.
--
      if ha_altitude >= pt_altitude then
          if ha_altitude - pt_altitude >= FAA_msd then
              case 3 * zero_test(ha_rate) + zero_test(pt_rate) is
                  when 2 | 5 | 8 =>
begin
                      change_rate := 60.0 * (FAA_msd - msd) /

float(Situation_Dynamics.get_elapsed_time(threat));
                      concatenate(msg_3, msg_suffix, change_rate);
exception
   when numeric_error => text_io.put_line("cam 2 - NE");
   when constraint_error => text_io.put_line("cam 2 - CE");
   when others => text_io.put_line("cam 2 - Bozo error");
end;

                  when 3 | 4 =>
                      Air_Traffic_Display_Device.write_text(msg_2, x_msg, y_msg);
                  when 6 =>
                      Air_Traffic_Display_Device.write_text(msg_1, x_msg, y_msg);
                  when others =>
                      null;
              end case;
          else
begin
              change_rate := 60.0 * (FAA_msd - msd) /

float(Situation_Dynamics.get_elapsed_time(threat));
              concatenate(msg_3, msg_suffix, change_rate);
exception
   when numeric_error => text_io.put_line("cam 3 - NE");
   when constraint_error => text_io.put_line("cam 3 - CE");
   when others => text_io.put_line("cam 3 - Bozo error");
```

```
end;
            end if;
        else
            if pt_altitude - ha_altitude >= FAA_msd then
                case 3 * zero_test(ha_rate) + zero_test(pt_rate) is
                    when 1 | 4 =>
begin
                        change_rate := 60.0 * (FAA_msd - msd) /

float(Situation_Dynamics.get_elapsed_time(threat));
                        concatenate(msg_4, msg_suffix, change_rate);
exception
    when numeric_error => text_io.put_line("cam 4 - NE");
    when constraint_error => text_io.put_line("cam 4 - CE");
    when others => text_io.put_line("cam 4 - Bozo error");
end;
                    when 3 =>
                        Air_Traffic_Display_Device.write_text(msg_1, x_msg, y_msg);
                    when 6 | 7 | 8 =>
                        Air_Traffic_Display_Device.write_text(msg_2, x_msg, y_msg);
                    when others =>
                        null;
                end case;
            else
begin
                change_rate := 60.0 * (FAA_msd - msd) /

float(Situation_Dynamics.get_elapsed_time(threat));
                concatenate(msg_4, msg_suffix, change_rate);
exception
    when numeric_error => text_io.put_line("cam 5 - NE");
    when constraint_error => text_io.put_line("cam 5 - CE");
    when others => text_io.put_line("cam 5 - Bozo error");
end;
            end if:
        end if;
exception
    when numeric_error => text_io.put_line("cam - NE");
    when constraint_error => text_io.put_line("cam - CE");
    when others => text_io.put_line("cam - Bozo error");
    end corrective_action_msg;


--
-- Update the icon shape for a potential_threat when its
-- partition changes.
--
    procedure update_ads(threat : in Potential_Threat.pt_handle)
    is
    begin
--   TBD
        null;
    end update_ads;
```

```ada
--
-- Initialize the air traffic display. Also position the host
-- aircraft icon in the center of the display as well. For
-- now, the icon shape is fixed to be a square.
--
   procedure initialize_display
   is
   begin
      Air_Traffic_Display_Device.create_display(xloc => x_origin,
                                                yloc => y_origin,
                                                width => x_size,
                                                height => y_size);
      host_aircraft_handle := Air_Traffic_Display_Device.create_object(
                         icon_shape        =>
Air_Traffic_Display_Device.square,
                         icon_size         => icon_size,
                         icon_fill         =>
Air_Traffic_Display_Device.black,
                         icon_color        =>
Air_Traffic_Display_Device.white,
                         fill_blink_rate => 0.0,
                         obj_blink_rate  => 0.0,
                         xloc              => x_center - icon_size / 2,
                         yloc              => y_center - icon_size / 2,
                         label_1           => " ",
                         label_2           => " ",
                         label_3           => " ",
                         label_4           => " ",
                         label_5           => " ");
   end initialize_display;


--
-- Update the information shown on the display for the
-- specified potential threat.
--
   procedure update_cws(threat : in Potential_Threat.pt_handle;
                        display_status : in Potential_Threat.target_display)
   is
      new_xloc : Air_Traffic_Display_Device.position;
      new_yloc : Air_Traffic_Display_Device.position;
      x_location, y_location : Physical_Quantities.nautical_mile;
      handle : Air_Traffic_Display_Device.display_handle;
   begin
--
-- If the display_status indicates that the target should be
-- removed from the display, then do it.
--
      if display_status = Potential_Threat.delete then
         Air_Traffic_Display_Device.delete_object(
                        Potential_Threat.get_display_handle(threat));
         return;
      end if;
--
```

```
-- Otherwise, we need to update the display. So, compute
-- the location on display (in terms of pixels) for
-- the potential threat. The new pixel location is determined by
-- converting the range from the potential threat to the host
-- aircraft into equivalent pixels. One pixel equals:
--
--      y_size / 300.0
--
-- where 300.0 is the canned surveillance area and 'y_size' is defined above.
--
      x_location := Potential_Threat.get_range(threat) *

Numerical_Algorithms.sin(Potential_Threat.get_relative_bearing(threat));
      y_location := Potential_Threat.get_range(threat) *

Numerical_Algorithms.cos(Potential_Threat.get_relative_bearing(threat));
      new_xloc := x_center + Air_Traffic_Display_Device.position(x_location *
                                                      (float(y_size) /
300.0));
      new_yloc := y_center - Air_Traffic_Display_Device.position(y_location *
                                                      (float(y_size) /
300.0));
--
-- Adjust these locations to the center of the icon.
--
      new_xloc := new_xloc - icon_size / 2;
      new_yloc := new_yloc - icon_size / 2;
--
-- Having the new position, determine whether we need to create an
-- icon for this potential threat or to simply move an existing
-- one. In either case, we must also update the host aircraft
-- attribute information.
--
      handle := Potential_Threat.get_display_handle(threat);
      if handle /= Air_Traffic_Display_Device.null_display_handle then
         Air_Traffic_Display_Device.move_object(
              id      => handle,
              xloc    => new_xloc,
              yloc    => new_yloc,
              label_1 => "ID: " & Potential_Threat.get_aircraft_id(threat),
              label_2 => "Altitude: " &
integer'image(integer(Potential_Threat.get_altitude(threat))),
              label_3 => "Airspeed: " &
integer'image(integer(Potential_Threat.get_velocity(threat))),
              label_4 => "Course: " &
integer'image(integer(Potential_Threat.get_ground_track(threat))),
              label_5 => "Range: " &
nautical_mile_to_string(Potential_Threat.get_range(threat)));
      else
         handle := Air_Traffic_Display_Device.create_object(
              icon_shape      => Air_Traffic_Display_Device.triangle,
              icon_size       => icon_size,
```

```
                icon_fill       => Air_Traffic_Display_Device.black,
                icon_color   ,  => Air_Traffic_Display_Device.white,
                fill_blink_rate => 0.0,
                obj_blink_rate  => 0.0,
                xloc            => new_xloc,
                yloc            => new_yloc,
                label_1         => "ID: " &
Potential_Threat.get_aircraft_id(threat),
                label_2         => "Altitude: " &
integer'image(integer(Potential_Threat.get_altitude(threat))),
                label_3         => "Airspeed: " &
integer'image(integer(Potential_Threat.get_velocity(threat))),
                label_4         => "Course: " &
integer'image(integer(Potential_Threat.get_ground_track(threat))),
                label_5         => "Range: " &
nautical_mile_to_string(Potential_Threat.get_range(threat)));
            Potential_Threat.set_display_handle(threat, handle);
        end if;
--
-- Update host aircraft information
--
        Air_Traffic_Display_Device.move_object(
            id       => host_aircraft_handle,
            xloc     => x_center - icon_size / 2,
            yloc     => y_center - icon_size / 2,
            label_1  => " ",
            label_2  => "Altitude: " &
integer'image(integer(Host_Aircraft.get_altitude)),
            label_3  => "Airspeed: " &
integer'image(integer(Host_Aircraft.get_velocity)),
            label_4  => "Course: " &
integer'image(integer(Host_Aircraft.get_ground_track)),
            label_5  => " ");
exception
    when constraint_error =>
        text_io.put_line("update_cws CE");
        text_io.put_line("Target info: " &
Potential_Threat.get_aircraft_id(threat));
        text_io.put("R: "); put(float(Potential_Threat.get_range(threat)), aft
=> 1,
                                                        exp => 0);
text_io.new_line;
        text_io.put("RB: ");
put(float(Potential_Threat.get_relative_bearing(threat)),
                                        aft => 1, exp => 0);
text_io.new_line;
    when numeric_error =>
        text_io.put_line("update_cws NE");
        text_io.put_line("Target info: " &
Potential_Threat.get_aircraft_id(threat));
        text_io.put("R: "); put(float(Potential_Threat.get_range(threat)), aft
=> 1,
```

```
                                                          exp => 0);
    text_io.new_line;
        text_io.put("RB: ");
    put(float(Potential_Threat.get_relative_bearing(threat)),
                                          aft => 1, exp => 0);

    text_io.new_line;
       when others =>
          text_io.put_line("update_cws BOZO error");
       end update_cws;

    end Air_Traffic_Display;
```

5. Audible_Alarm (AA)

Spec

```
--
-- Audible_Alarm (AA)
--
-- This module determines the frequency and duration at which
-- to ring the audible alarm for a specified collision warning
-- situation.
--
with Potential_Threat;
package Audible_Alarm is

    procedure ring_alarm(cws : in Potential_Threat.cws_id);

end Audible_Alarm;
```

Body

```
{
^module!external(aa_body)

^type(ring_info, (cws_name : target,
                  frequency : target,
                  duration : target))

^program(aa, (ring : list of ring_info))

^module!internal(aa_body)

^prog_impl(aa, body, (}
--
-- Audible_Alarm (AA) body
--
-- The audible_alarm device generates a tone that can be heard
-- within the host_aircraft cockpit.
--
with Potential_Threat;
with Audible_Alarm_Device;
package body Audible_Alarm is

    procedure ring_alarm(cws : in Potential_Threat.cws_id)
    is
```

```
    begin
        case cws is
{^forall(r, ring, ()
            when Potential_Threat.{r.cws_name} =>
                Audible_Alarm_Device.ring_alarm(f => {r.frequency}, d =>
{r.duration});
{))}
        when others =>
            return;
        end case;
    end ring_alarm;

end Audible_Alarm;
{))}
```

6. Audible_Alarm_Device (AAD)

Spec

```
{
^program(aad, (loosely_coupled : target))

^prog_impl(aad, v1, ()
--
-- Audible_Alarm_Device (AAD) spec
--
-- The audible_alarm device generates a tone that can be heard
-- within the host_aircraft cockpit.
--
package Audible_Alarm_Device is

   type Duration is delta 0.01 range 0.01 .. 10.00;     -- seconds
   type Frequency is range 1000 .. 10_000;              -- hertz

   procedure ring_alarm(f : in Frequency;
                        d : in Duration);
{^select(
   ^equal(loosely_coupled, {True}) -> ()

   type Alarm_Message_Type is private;

private
   type Alarm_Message_Type is
      record
         Frequency : Frequency;
         Duration : Duration;
      end record;
{))}

end Audible_Alarm_Device;
{))}
```

Body

TBD

7. Collision_Warning_Situation_Status (CWSS)

Spec

```
--
-- Collision Warning Situation Status (CWSS) spec
--
-- This module determines the collision warning situation status
-- for the given potential threat and host aircraft.
--
with Potential_Threat;
package Collision_Warning_Situation_Status is

    function determine_cws_status(threat : in Potential_Threat.pt_handle)
                                        return Potential_Threat.cws_id;

    function determine_host_cws_status return Potential_Threat.cws_id;

end Collision_Warning_Situation_Status;
```

Body

```
{
^module!external (cwss_body)

^type(time_type, (min : target,
                  max : target))

^type(range_type, (min : target,
                   max : target))

^type(t_and_r_type, (t_min : target,
                     t_max : target,
                     r_min : target,
                     r_max : target))

^type(cws_def, (time ?: time_type,
                range ?: range_type,
                t_and_r ?: t_and_r_type))

^type(cws_type, (cws_name : target,
                 severity : target,
                 predicate : cws_def,
                 partition : target))

^program(cwss, (cws : list of cws_type))

^module!internal(cwss_body)

^prog_impl(cwss, body, (}
--
-- Collision Warning Situation Status (CWSS)  package body
--
-- This module determines the collision warning situation status
-- for the given potential_threat and host_aircraft.
```

```
--
with Potential_Threat;
{^select(
    ^member(^filter(x, cws, ^not(^equal(x.partition, {ALL}))))) -> ()
with PT_Partition; use PT_Partition;
{)))}
with Physical_Quantities; use Physical_Quantities;
{^select(
    ^member(^filter(x, cws,
                ^or(^defined(x.predicate.time), ^defined(x.predicate.t_and_r))))
-> ()
with Situation_Dynamics;
with Text_IO;
{)))}
package body Collision_Warning_Situation_Status is

--
-- This routine keeps track of the number of potential
-- threats in each collision situation. This enables us to
-- quickly determine the host aircraft status when
-- requested to provide it.
--
    target_count : array(Potential_Threat.cws_id'first ..
                            Potential_Threat.cws_id'last) of integer := (others
=> 0);

--
-- Determine the collision warning situation status of the specified
-- potential threat.
--
    function determine_cws_status(threat : in Potential_Threat.pt_handle)
                                        return Potential_Threat.cws_id
    is
        airspeed_and_altitude_valid : boolean;
{^select(
    ^member(^filter(x, cws, ^not(^equal(x.partition, {ALL}))))) -> ()
        partition : Potential_Threat.partition;
{))
^select(
    ^member(^filter(x, cws,
                ^or(^defined(x.predicate.time), ^defined(x.predicate.t_and_r))))
-> ()
        time_to_intersect : Physical_Quantities.seconds;
{))
^select(
    ^member(^filter(x, cws,
                ^or(^defined(x.predicate.range),
^defined(x.predicate.t_and_r))))) -> ()
        target_range : Physical_Quantities.nautical_mile;
{)))}
        old_cws_status, new_cws_status : Potential_Threat.cws_id;
    begin
        airspeed_and_altitude_valid := Potential_Threat.altitude_valid(threat)
```

```
and then

Potential_Threat.velocity_valid(threat);
{^select(
    ^member(^filter(x, cws, ^not(^equal(x.partition, {ALL})))) -> ()
        partition := PT_Partition.get_partition(threat);
())
^select(
    ^member(^filter(x, cws,
                ^or(^defined(x.predicate.range),
^defined(x.predicate.t_and_r)))) -> ()
        target_range := Potential_Threat.get_range(threat);
())
^select(
    ^member(^filter(x, cws,
                ^or(^defined(x.predicate.time), ^defined(x.predicate.t_and_r))))
-> ()
        if (airspeed_and_altitude_valid) then
            time_to_intersect := Situation_Dynamics.get_elapsed_time(threat);
        end if;
())}
        old_cws_status := Potential_Threat.get_cws_status(threat);
        if (
{^forall(c, cws, (
    ^select(
        ^not(^equal(c.partition, {ALL})) -> ()
            partition = Potential_Threat.{c.partition} and then
())
    ^select(
        ^defined(c.predicate.range) -> ()
            ({c.predicate.range.min} <= target_range and then target_range <
{c.predicate.range.max})) then
                new_cws_status := Potential_Threat.{c.cws_name};
{      )
        ^defined(c.predicate.time) -> ()
            (airspeed_and_altitude_valid) and then
            ({c.predicate.time.min} <= time_to_intersect and then
                        time_to_intersect < {c.predicate.time.max})) then
                new_cws_status := Potential_Threat.{c.cws_name};
{      )
        ^defined(c.predicate.t_and_r) -> ()
            (airspeed_and_altitude) and then
            ((({c.predicate.t_and_r.r_min} <= target_range and then target_range <

                                                {c.predicate.t_and_r.r_max})
            or else
            ({c.predicate.t_and_r.t_min} <= time_to_intersect and then
                        time_to_intersect < {c.predicate.t_and_r.t_max}))) then
            new_cws_status := Potential_Threat.{c.cws_name};
{      )
    )
    ^select(
```

```
          ^not(^last(c)) -> (}
          elsif (
{       )
     )
))}
          else
             new_cws_status := Potential_Threat.normal;
          end if;
          if (target_count(old_cws_status) /= 0) then
             target_count(old_cws_status) := target_count(old_cws_status) - 1;
          end if;
          target_count(new_cws_status) := target_count(new_cws_status) + 1;
          return new_cws_status;
exception
    when constraint_error => text_io.put_line("determine cws CE"); return
Potential_Threat.normal;
    when numeric_error => text_io.put_line("determine cws NE"); return
Potential_Threat.normal;
    when others => text_io.put_line("determine cws Bozo error"); return
Potential_Threat.normal;
    end determine_cws_status;


--
-- Determine the collision warning situation status of
-- the host aircraft. Each the number of potential threats
-- in each situation category starting with the most severe
-- situation and progressing to the least severe. The
-- first collision warning situation encountered which has
-- a non-zero target count is the status of the host aircraft.
-- If all situations have zero potential threats, then the
-- status of the host aircraft is "normal".
--
    function determine_host_cws_status return Potential_Threat.cws_id
    is
    begin
        if (
{^forall(c, cws, (}
    target_count(Potential_Threat.{c.cws_name}) /= 0) then
        return Potential_Threat.{c.cws_name};
{ ^select(
      ^not(^last(c)) -> (}
      elsif (
{     )
    )
))}
        else
           return Potential_Threat.normal;
        end if;
    end determine_host_cws_status;


end Collision_Warning_Situation_Status;
()))}
```

8. Communication (COMM)

Spec

```
{
^module!external(comm_spec)

^type(atc_info, (cws_name : target,
                 code : target))

^type(inter_air_info, (cws_name : target,
                       code : target))

^program(comm, (atc_msg : list of atc_info,
                inter_air_msg : list of inter_air_info,
                mode : target))
}

{
^module!internal(comm_spec)

^prog_impl(comm, spec, (}
--
-- Communication (COMM)  package spec
--
-- This module determines the content of the ATC_Msg and
-- Inter_Air_Msg messages that are transmitted to the
-- air traffic control center and potential threat, respectively for
-- a specified collision warning situation.
--
with Potential_Threat;
package Communication is
{^select(
    ^member(atc_msg) -> (}

    procedure send_atc_msg(cws : in Potential_Threat.cws_id);
{))
^select(
    ^member(inter_air_msg) -> (}

    procedure send_ia_msg(cws : in Potential_Threat.cws_id);
{))}

end Communication;
{))}
```

Body

```
{
^module!external(comm_body)

^type(atc_info, (cws_name : target,
                 code : target))
```

```
^type(inter_air_info, (cws_name : target,
                       code : target))

^program(comm, (atc_msg : list of atc_info,
                inter_air_msg : list of inter_air_info.
                mode : target))
}

{
^module!internal(comm_body)

^prog_impl(comm, body, (
{
--
-- Communication (COMM)  package body
--
with Potential_Threat;
with Communication_Device;}
^select(
   ^or(^member(inter_air_msg), ^equal(mode, {C})) -> ( {
with Host_Aircraft;}
   )
)
^select(
   ^member(inter_air_msg) -> ( {
with Physical_Quantities;}
   )
)
{
package body Communication is}
^select(
   ^member(atc_msg) -> ( {

   procedure send_atc_msg(cws : in Potential_Threat.cws_id)
   is
   begin
      case cws is}
^forall(c, atc_msg. ( {

      when Potential_Threat.{c.cws_name} =>
          Communication_Device.send_atc_msg(code => {c.code}}
   ^select(
      ^equal(mode, {C}) -> ( {
             , altitude => Host_Aircraft.get_altitude}
      )
   ;
{
            );
}
))
{
      when others =>
          return;
      end case;
```

```
    end send_atc_msg;}
    )
)
^select(
    ^member(inter_air_msg) -> ( {

    procedure send_ia_msg(cws : in Potential_Threat.cws_id)
    is
        latitude : Physical_Quantities.degrees;
        longitude : Physical_Quantities.degrees;
    begin
        Host_Aircraft.get_location(latitude => latitude,
                                   longitude => longitude);
        case cws is}
^forall(i, inter_air_msg, ( {

        when Potential_Threat.{i.cws_name} =>
            Communication_Device.send_ia_msg(code => {i.code},
                                              altitude =>
Host_Aircraft.get_altitude,

                                              latitude => latitude,
                                              longitude => longitude);}
))
{
        when others =>
            return;
        end case;
    end send_ia_msg;}
    )
)
{

end Communication;
}
))
}
```

9. Communication_Device (CD)

Spec

```
{
^module!external(cd_spec)

^program(variant, (atc_msg: target,
                   inter_air_msg : target))

^prog_impl(variant, v1, (
    ^select(
        ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True}), -> ( {{}
(msg : Msg_Type){}
})))
))
```

```
^program(inter_air_msg_part, ())

^prog_impl(inter_air_msg_part, v1, ( {
   altitude  : Physical_Quantities.feet;
   latitude  : Physical_Quantities.degrees;
   longitude : Physical_Quantities.degrees;
}))

^program(cd, (atc_msg : target,
              inter_air_msg : target,
              mode : target,
              loosely_coupled : target))
}

{
^module!internal(cd_spec)

^prog_impl(cd, spec, (
{
--
-- Communication_Device (CD)  package spec
--
-- This module encapsulates the hardware / software
-- interface to the communication device. It knows how
-- to transmit a message to either an air traffic
-- control center or to a specified potential threat.
--
with Physical_Quantities;
package Communication_Device is
}
^select(
   ^equal(atc_msg, {True}) -> ( {
--
-- Send the ATC_Msg to the nearest air traffic control center.
--
   procedure send_atc_msg(code : in natural {}
}
   ^select(
      ^equal(mode, {C}) -> ( {
                                          ;
                        altitude : in Physical_Quantities.feet
}))
{
                              );
}))

^select(
   ^equal(inter_air_msg, {True}) -> ( {
--
-- Send an Inter-Air_Msg to the specified potential_threat.
--
   procedure senu_inter_air_msg(code : in natural;
                                altitude : in Physical_Quantities.feet;
                                latitude : in Physical_Quantities.degrees;
```

```
                                            longitude : in Physical_Quantities.degrees);
}))

^select(
   ^equal(loosely_coupled, {True}) -> (
      ^select(
         ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True})) -> ( {
   type Msg_Type is private;
}))
{
   type Communication_Msg_Type}^variant(atc_msg, inter_air_msg){ is private;

private
}
   ^select(
      ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True})) -> ( {
   type Msg_Type is (ATC, Inter_Air);
}))
{
   type Communication_Msg_Type}^variant(atc_msg, inter_air_msg){ is
      record
         code : natural;
}
         ^select(
            ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True})) -> (
{
               case msg is
               when ATC =>
}
                  ^select(
                     ^equal(mode, {C}) -> ( {
                        altitude : Physical_Quantities.feet;
})
                     true -> ( {
                        null;
}))
{              when Inter_Air => }
^inter_air_msg_part()
{
               end case;

})
            ^equal(atc_msg, {True}) -> (
               ^select(
                  ^equal(mode, {C}) -> ( {
                     altitude : Physical_Quantities.feet;
}))
)
            ^equal(inter_air_msg, {True}) -> (
               ^inter_air_msg_part()
))
{
      end record;
```

```
}))
{
end Communication_Device;
}))
}
```

Body

```
{
^module!external(cd_body)

^program(variant, (atc_msg: target,
                    inter_air_msg : target))

^prog_impl(variant, v1, (
   ^select(
       ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True})) -> ( {{}
(msg : Msg_Type){}
}))
))

^program(inter_air_msg_part, ())

^prog_impl(inter_air_msg_part, v1, ( {
   altitude : Physical_Quantities.feet;
   latitude : Physical_Quantities.degrees;
   longitude : Physical_Quantities.degrees;
}))

^program(cd, (atc_msg : target,
              inter_air_msg : target,
              mode : target,
              loosely_coupled : target))
}

{
^module!internal(cd_body)

^prog_impl(cd, body, (
{
--
-- Communication_Device (CD)  package body
--
}
^select(
   ^equal(loosely_coupled, {True}) -> ( {
with Communication_Buffer;
}))
{with Physical_Quantities;
with System;
package body Communication_Device is
}
^select(
   ^equal(loosely_coupled, {True}) -> ( {
```

```
    task output_communication is
       pragma Priority(10);
    end output_communication;

    task body output_communication is
       message : Communication_Msg_Type;
    begin
       loop
          Communication_Buffer.Receive(message);
--        write_to_physical_device(??);
       end loop;
    end output_communication;
}))
^select(
    ^equal(atc_msg, {True}) -> ( {
--
-- Send the ATC_Msg to the nearest air traffic control center.
--
    procedure send_atc_msg(code : in natural
}
    ^select(
       ^equal(mode, {C}) -> ( {

                              altitude : in Physical_Quantities.feet
}))
{
                       )
    is
}
    ^select(
       ^equal(loosely_coupled, {True}) -> ( {
       message : Communication_Msg_Type}
       ^select(
          ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True})) -> ( {
             (msg => ATC) {}
}))
{;

}))
{  begin
}
    ^select(
       ^equal(loosely_coupled, {True}) -> ( {
       message.code := code;}
       ^select(
          ^equal(mode, {C}) -> ( {
       message.altitude := altitude;
}))
{
       Communication_Buffer.send(message);
})
       true -> ( {
--     write_to_physical_device();
```

```
})))
{
   end send_atc_msg;
}))

^select(
   ^equal(inter_air_msg, {True}) -> ( {
--
-- Send an Inter-Air_Msg to the specified potential_threat.
--
   procedure send_inter_air_msg(code : in natural;
                                altitude : in Physical_Quantities.feet;
                                latitude : in Physical_Quantities.degrees;
                                longitude : in Physical_Quantities.degrees)
   is
}
   ^select(
      ^equal(loosely_coupled, {True}) -> ( {
      message : Communication_Msg_Type}
      ^select(
         ^and(^equal(atc_msg, {True}), ^equal(inter_air_msg, {True})) -> ({
            (msg => Inter_Air) {}
})))
{;
})))
{  begin
}
      ^select(
         ^equal(loosely_coupled, {True}) -> ( {
         message.code := code;
         message.altitude := altitude;
         message.latitude := latitude;
         message.longitude := longitude;
         Communication_Buffer.send(message);
})
         true -> ( {
--       write_to_physical_device();
})))
{
   end send_inter_air_msg;

})))
{end Communication_Device;}

))
}
```

10. Host_Aircraft (HA)

Spec

```
--
-- Host_Aircraft (HA) package spec
--
```

```
-- This module models the host aircraft in an ATD/CWM system. The
-- host aircraft has properties of altitude, aircraft_identification,
-- airspeed, location, ground_track, climb rate, and cws_status. The
-- hidden decisions of this module are the internal representation
-- of these properties, algorithms for manipulating them, and how to
-- determine the values for these properties.
--
with Physical_Quantities;
with Potential_Threat;
package Host_Aircraft is


--
-- Returns the most recently measured altitude of the host aircraft.
--
    function get_altitude return Physical_Quantities.feet;


--
-- Returns the most recently measured climb rate of the host aircraft.
--
    function get_climb_rate return Physical_Quantities.fpm;


--
-- Returns the collision warning situation status of the host aircraft.
--
    function get_cws_status return Potential_Threat.cws_ia;


--
-- Returns the most recently measured ground track of the host aircraft.
--
    function get_ground_track return Physical_Quantities.degrees;


--
-- Returns the most recent values for all properties of the host aircraft.
--
    procedure get_host_data(altitude : out Physical_Quantities.feet;
                            ground_track : out Physical_Quantities.degrees;
                            rate : out Physical_Quantities.fps;
                            airspeed : out Physical_Quantities.knots;
                            latitude : out Physical_Quantities.latitude;
                            longitude : out Physical_Quantities.longitude;
                            status : out Potential_Threat.cws_id);


--
-- Returns the most recently measured position of the host aircraft
--
    procedure get_location(latitude : out Physical_Quantities.latitude;
                           longitude : out Physical_Quantities.longitude);


--
-- Returns the most recently measured airspeed of the host aircraft.
--
    function get_velocity return Physical_Quantities.knots;

end Host_Aircraft;
```

Body

```
--
-- Host_Aircraft (HA)  package body
--
-- This module models the host aircraft in an ATD/CWM system. The
-- host aircraft has properties of altitude, aircraft_identification,
-- airspeed, location, ground_track, climb rate, and cws_status. The
-- hidden decisions of this module are the internal representation
-- of these properties, algorithms for manipulating them, and how to
-- determine the values for these properties.
--
with Physical_Quantities;
with Potential_Threat;
with Navigation;
with Collision_Warning_Situation_Status;
with Air_Craft_Motion;
with System;
package body Host_Aircraft is


--
-- Constants
--
   Host_Aircraft_Update_Frequency : constant := 1.0;


--
-- Information block for the host_aircraft.
--
   type host_aircraft_info is
      record
         altitude_Y : Physical_Quantities.feet; -- most recent altitude
reading
         time_Y : Physical_Quantities.seconds;
         altitude_X : Physical_Quantities.feet; -- previous altitude reading
         time_X : Physical_Quantities.seconds;
         velocity : Physical_Quantities.knots;
         climb_rate : Physical_Quantities.fpm;
         latitude : Physical_Quantities.latitude;
         longitude : Physical_Quantities.longitude;
         ground_track : Physical_Quantities.degrees;
         cws_status : Potential_Threat.cws_id;
      end record;

   host_aircraft : host_aircraft_info := (
      altitude_Y => 0.0,
      time_Y => 0.0,
      altitude_X => 0.0,
      time_X => 0.0,
      velocity => 0.0,
      climb_rate => 0.0,
      latitude => 0.0,
      longitude => 0.0,
      ground_track => 0.0,
```

```
            cws_status => Potential_Threat.normal
    );


    --
    -- Returns the most recently measured altitude of the host aircraft.
    --
    function get_altitude return Physical_Quantities.feet
    is
    begin
       return host_aircraft.altitude_Y;
    end get_altitude;


    --
    -- Returns the most recently measured climb rate of the host_aircraft.
    --
    function get_climb_rate return Physical_Quantities.fpm
    is
    begin
       return host_aircraft.climb_rate;
    end get_climb_rate;


    --
    -- Returns the collision warning situation status of the host_aircraft.
    --
    function get_cws_status return Potential_Threat.cws_id
    is
    begin
       return host_aircraft.cws_status;
    end get_cws_status;


    --
    -- Returns the most recently measured ground track of the host_aircraft.
    --
    function get_ground_track return Physical_Quantities.degrees
    is
    begin
       return host_aircraft.ground_track;
    end get_ground_track;


    --
    -- Returns the most recent values for all properties of the host_aircraft.
    --
    procedure get_host_data(altitude : out Physical_Quantities.feet;
                            ground_track : out Physical_Quantities.degrees;
                            rate : out Physical_Quantities.fps;
                            airspeed : out Physical_Quantities.knots;
                            latitude : out Physical_Quantities.latitude;
                            longitude : out Physical_Quantities.longitude;
                            status : out Potential_Threat.cws_id)
    is
    begin
       altitude := host_aircraft.altitude_Y;
       ground_track := host_aircraft.ground_track;
       rate := host_aircraft.climb_rate;
```

```
         airspeed := host_aircraft.velocity;
         latitude := host_aircraft.latitude;
         longitude := host_aircraft.longitude;
         status := host_aircraft.cws_status;
     end get_host_data;


--
-- Returns the most recently measured position of the host_aircraft.
--

     procedure get_location(latitude : out Physical_Quantities.latitude;
                            longitude : out Physical_Quantities.longitude)
     is
     begin
         latitude := host_aircraft.latitude;
         longitude := host_aircraft.longitude;
     end get_location;


--
-- Returns the most recently measured airspeed of the host_aircraft.
--

     function get_velocity return Physical_Quantities.knots
     is
     begin
         return host_aircraft.velocity;
     end get_velocity;


--
-- Task to retrieve navigation data on the host aircraft from
-- the navigation device on a periodic basis since this device
-- is passive. The periodicity is given by Host_Aircraft_Update_Frequency.
--

     task update_host_aircraft_information is
         pragma Priority(6);
     end update_host_aircraft_information;

     task body update_host_aircraft_information
     is
     begin
         delay 7.0;    -- For simulation PURPOSES only to allow X interface setup
         loop
            delay Host_Aircraft_Update_Frequency;
            host_aircraft.altitude_X := host_aircraft.altitude_Y;
            host_aircraft.time_X := host_aircraft.time_Y;
            Navigation.get_nav_data(host_aircraft.altitude_Y,
                                    host_aircraft.time_Y,
                                    host_aircraft.velocity,
                                    host_aircraft.ground_track,
                                    host_aircraft.latitude,
                                    host_aircraft.longitude);

--
-- Compute the cws_status and climb rate as well.
--

            host_aircraft.cws_status :=
```

```
Collision_Warning_Situation_Status.determine_host_cws_status;
        host_aircraft.climb_rate :=
                Air_Craft_Motion.get_climb_rate(host_aircraft.altitude_Y,
                                        host_aircraft.time_Y,
                                        host_aircraft.altitude_X,
                                        host_aircraft.time.X);
    end loop;
  end update_host_aircraft_information;

end Host_Aircraft;
```

11. Initialization_and_Termination (IT)

Body

```
--
-- The "main" procedure for the ATD/CWM system. It
-- starts all the tasks in the system and causes the
-- air traffic display to be initialized.
--
with Air_Traffic_Display;
with System;
pragma Elaborate(Air_Traffic_Display);
procedure Atd_Cwm
is
    pragma Priority(15);
begin
--
-- Initialize the air traffic display.
--
    Air_Traffic_Display.initialize_display;
    loop
        delay 86_400.0;
    end loop;
end Atd_Cwm;
```

12. Navigation (NAV)

Spec

```
--
-- Navigation (NAV)    package spec
--
-- This module encapsulates the hardware / software interface to the
-- host aircraft navigation device. The primary hidden decisions are
-- how to obtain host aircraft raw data for altitude, airspeed, ground
-- track, latitude, and longitude; the scale and format of these input
-- data items; and the device-dependent operations that must be
-- applied to convert the raw data to the internal format of the
-- ATD/CWM system.
--
with Physical_Quantities;
package Navigation is
```

```
        procedure get_nav_data(altitude : out Physical_Quantities.feet;
                               timestamp : out Physical_Quantities.seconds;
                               airspeed : out Physical_Quantities.knots;
                               ground_track : out Physical_Quantities.degrees;
                               latitude : out Physical_Quantities.latitude;
                               longitude : out Physical_Quantities.longitude);

    end Navigation;
```

Body

```
    --
    -- Navigation (NAV)    package body
    --
    -- This module encapsulates the hardware / software interface to the
    -- host aircraft navigation device. The primary hidden decisions are
    -- how to obtain host aircraft raw data for altitude, airspeed, ground
    -- track, latitude, and longitude; the scale and format of these input
    -- data items; and the device-dependent operations that must be
    -- applied to convert the raw data to the internal format of the
    -- ATD/CWM system.
    --
    with Physical_Quantities;
    with Simulation_Data;
    package body Navigation is

    --
    -- Read the host aircraft navigation data ad return the converted
    -- information to the calling program.
    --
        procedure get_nav_data(altitude : out Physical_Quantities.feet;
                               timestamp : out Physical_Quantities.seconds;
                               airspeed : out Physical_Quantities.knots;
                               ground_track : out Physical_Quantities.degrees;
                               latitude : out Physical_Quantities.latitude;
                               longitude : out Physical_Quantities.longitude)
        is
        begin
    --
    -- Get information from navigation "device".
    --
            Simulation_Data.get_sim_data(altitude, airspeed, ground_track,
                                                  latitude, longitude);
            timestamp := Physical_Quantities.get_time;
        end get_nav_data;

    end Navigation;
```

13. Numerical_Algorithms (NA)

NOTE: The package Spec and Body for this component have been purposely omitted to reduce the size of the ATD/CWM case study documentation.

14. Physical_Quantities (PQ)

NOTE: The package Spec and Body for this component have been purposely omitted to reduce the size of the ATD/CWM case study documentation.

15. Potential_Threat (PT)

Spec

```
{
^module!external(pt_spec)

^type(time_type, (min : target,
                  max : target))

^type(range_type, (min : target,
                   max : target))

^type(t_and_r_type, (t_min : target,
                     t_max : target,
                     r_min : target,
                     r_max : target))

^type(cws_def, (time ?: time_type,
                range ?: range_type,
                t_and_r ?: t_and_r_type))

^type(cws_info, (cws_name : target,
                 severity : target,
                 predicate : cws_def,
                 partition : target,
                 alarm : target,
                 atc_msg : target,
                 inter_air_msg : target,
                 corrective : target))

^program(pt, (cws : list of cws_info))

^module!internal(pt_spec)

^prog_impl(pt, spec, (}
--
-- Potential_Threat (PT)  package spec
--
-- This module models potential threats in an ATD/CWM system. Potential
-- threats have properties of altitude, airspeed, aircraft_identification,
-- ground_track, range, relative_bearing, climb_rate, and collision
-- warning situation status. This module knows how to
-- determine values for these properties.
--
with Physical_Quantities;
with Air_Traffic_Display_Device;
package Potential_Threat is

    type pt_handle is private;
```

```
    type target_source is (RADAR_SOURCE, ATC_SOURCE);
    type target_info(from : target_source := RADAR_SOURCE) is private;

    type target_display is (add_modify, delete);

    type partition is (ID, UID);    -- ID is identified
                                    -- UID is unidentified

    type cws_id is (
{^forall(c, cws, (}
     {c.cws_name},
{))}
     normal
    );

--
-- Returns whether the altitude value for the specified
-- potential threat is valid.
--
    function altitude_valid(threat : in pt_handle) return boolean;


--
-- Returns the aircraft identification of the specified potential_threat.
--
    function get_aircraft_id(threat : in pt_handle) return string;


--
-- Returns the current altitude of the specified potential_threat.
--
    function get_altitude(threat : in pt_handle) return
Physical_Quantities.feet;


--
-- Returns the most recent collision warning situation status of
-- the specified potential_threat.
--
    function get_cws_status(threat : in pt_handle) return cws_id:


--
-- Returns the current ground track of the specified potential_threat.
--
    function get_ground_track(threat : in pt_handle) return
Physical_Quantities.degrees;


--
-- Returns the value of the display handle for the specified potential threat.
--
    function get_display_handle(threat : in pt_handle)
                    return Air_Traffic_Display_Device.display_handle;


--
-- Sets the display handle for the specified potential threat.
--
    procedure set_display_handle(threat : in pt_handle;
```

```
                                         handle : in
   Air_Traffic_Display_Device.display_handle);


   --
   -- Returns which partition the potential_threat is a member of.
   --
      function get_partition(threat : in pt_handle) return partition;


   --
   -- Returns the most recently measured range between the specified
   -- potential_threat and the host_aircraft.
   --
      function get_range(threat : in pt_handle) return
   Physical_Quantities.nautical_mile;


   --
   -- Returns the most recently measured climb rate for the
   -- potential_threat.
   --
      function get_climb_rate(threat : in pt_handle) return
   Physical_Quantities.fpm;


   --
   -- Returns the most recently measured relative_bearing of the
   -- specified potential_threat.
   --
      function get_relative_bearing(threat : in pt_handle)
                                         return Physical_Quantities.degrees;


   --
   -- Returns the most recently measured velocity of the specified
   -- potential_threat.
   --
      function get_velocity(threat : in pt_handle) return
   Physical_Quantities.knots;


   --
   -- Returns a status which indicates whether the velocity of the
   -- specified potential threat is valid.
   --
      function velocity_valid(threat : in pt_handle) return boolean;

   private

      type pt_info;
      type pt_handle is access pt_info;

      type target_info(from : target_source := RADAR_SOURCE) is
         record
            aircraft_id : string(1..8);
            relative_bearing : Physical_Quantities.degrees;
            target_range : Physical_Quantities.nautical_mile;
            timestamp : Physical_Quantities.seconds;
            case from is
```

```
                when RADAR_SOURCE =>
                    sweep : integer;
                when ATC_SOURCE =>
                    altitude : Physical_Quantities.feet;
                    airspeed : Physical_Quantities.knots;
                    ground_track : Physical_Quantities.degrees;
            end case;
        end record;

    end Potential_Threat;
{)))
```

Body

```
{
^module!external(pt_body)

^type(time_type, (min : target,
                  max : target))

^type(range_type, (min : target,
                   max : target))

^type(t_and_r_type, (t_min : target,
                     t_max : target,
                     r_min : target,
                     r_max : target))

^type(cws_def, (time ?: time_type,
                range ?: range_type,
                t_and_r ?: t_and_r_type))

^type(cws_info, (cws_name : target,
                 severity : target,
                 predicate : cws_def,
                 partition : target,
                 alarm : target,
                 atc_msg : target,
                 inter_air_msg : target,
                 corrective : target))

^program(pt, (cws : list of cws_info))

^module!internal(pt_body)

^prog_impl(pt, body, (
{
--
-- Potential_Threat (PT)  package body
--
-- This module models potential threats in an ATD/CWM system. Potential
-- threats have properties of altitude, airspeed, aircraft_identification,
-- ground_track, range, relative_bearing, climb_rate, and collision
-- warning situation status. This module knows how to
```

```
-- determine values for these properties.
--
with Physical_Quantities;
} ^select(
      ^member(^filter(x, cws, ^equal(x.alarm, {True}))) -> ( {
with Audible_Alarm;
})){
with Air_Traffic_Display;
} ^select(
      ^or(^member(^filter(x, cws, ^equal(x.atc_msg, {True}))),
          ^member(^filter(c, cws, ^equal(x.inter_air_msg, {True})))) -> ( {
with Communication;
})){
with Air_Traffic_Control;
with Air_Traffic_Display_Device;
with Air_Craft_Motion;
with System:
with Target_Buffer;
with Collision_Warning_Situation_Status;
with Radar_Target_Priority_Buffer;
with Radar;
with Unchecked_Deallocation;
with Text_IO;
pragma Elaborate(Air_Traffic_Control, Target_Buffer):
pragma Elaborate(Radar_Target_Priority_Buffer, Radar);
package body Potential_Threat is


--
-- Information block for potential threats.
--
    type data_validity is (valid, invalid);
    type pt_info is
       record
          aircraft_id : string(1..8);
          altitude_Y : Physical_Quantities.feet; -- most recent altitude
reading
          altitude_Y_valid : data_validity;
          time_Y : Physical_Quantities.seconds;
          altitude_X : Physical_Quantities.feet; -- previous altitude reading
          time_X : Physical_Quantities.seconds;
          velocity : Physical_Quantities.knots;
          velocity_valid : data_validity;
          climb_rate : Physical_Quantities.fpm;
          ground_track : Physical_Quantities.degrees;
          cws_status : Potential_Threat.cws_id;
          target_range : Physical_Quantities.nautical_mile;
          relative_bearing : Physical_Quantities.degrees;
          handle : Air_Traffic_Display_Device.display_handle;
          sweep : integer;
       end record;


--
-- Various constants.
```

```
   --
   Purge_Time : constant := 5.0;
   Startup_Delay : constant := 10.0; -- for SIMULATION purposes only


   --
   -- Symbol table entries and symbol table. Size of symbol table
   -- given by HASHSIZE.
   --
   HASHSIZE : constant natural := 128;

   type table_entry;
   type next is access table_entry;

   type table_entry is
      record
         target : pt_handle;    -- potential threat
         link : next;           -- pointer to next entry on hash chain
      end record;

   buckets : array(0..HASHSIZE-1) of next := (others => null);


   --
   --
   -- Mapping of collision warning situations to priorities
   -- for the radar target priority buffer.
   --
   cws_to_priority : array(cws_id range cws_id'first ..}
   ^forall(c, cws, (
      ^select(
         ^last(c) -> ( { {c.cws_name} ) } )
      )
   )){

                           of Radar_Target_Priority_Buffer.message_priority :=
(}
   ^forall(c, cws, ( {
           {c.cws_name} => Radar_Target_Priority_Buffer.{c.cws_name}}
      ^select(
         ^not(^last(c)) -> ( {,}
      ))
   )){
           );


   --
   -- Procedures to deallocate storage previously allocated
   -- for the symbol table entries.
   --
   procedure free_table_entry is new Unchecked_Deallocation(table_entry,
next);
   procedure free_target is new Unchecked_Deallocation(pt_info, pt_handle);


   --
   -- Potential threat lookup routine. Lookup is based on the potential
   -- threat name. The potential threat information block is returned.
   --
```

```
    function lookup(name : in string) return pt_handle
    is
        hash_value : natural;
        ptr : next;
    begin
--
-- First, compute the hashing value for this target name.
--
        hash_value := 0;
        for n in name'first .. name'last loop
            hash_value := hash_value * 2 + character'pos(name(n));
        end loop;
        hash_value := hash_value rem HASHSIZE;
        ptr := buckets(hash_value);
        while ptr /= null loop
            if ptr.target.aircraft_id = name then
                return ptr.target;
            end if;
            ptr := ptr.link;
        end loop;
        return null;
    end lookup;


--
-- Install potential threat in the symbol table and return a
-- pointer to the target information block.
--
    function install(name : in string) return pt_handle
    is
        ptr : next;
        hash_value : natural;
    begin
        hash_value := 0;
        for n in name'first .. name'last loop
            hash_value := hash_value * 2 + character'pos(name(n));
        end loop;
        hash_value := hash_value rem HASHSIZE;
        ptr := new table_entry;
        ptr.target := new pt_info;
        ptr.target.aircraft_id := name;
        ptr.link := buckets(hash_value);
        buckets(hash_value) := ptr;
        return ptr.target;
    end install;


--
-- Background task to periodically purge old out-of-date
-- target information blocks. An 'old' information block
-- is any block that has not been updated within the
-- last Purge_Time seconds.
--
    task purge_target_information_blocks
    is
```

```
        pragma Priority(11);
    end purge_target_information_blocks;


    task body purge_target_information_blocks
    is
        ptr1, ptr2, ptr3 : next;
        current_time : Physical_Quantities.seconds;

    begin
        loop
            delay Purge_Time;
            current_time := Physical_Quantities.get_tin  ·
            for n in buckets'first .. buckets'last loop
                ptr1 := null;
                ptr2 := buckets(n);
                while ptr2 /= null loop
                    if current_time - ptr2.target.time_Y > Purge_Time then
                        ptr3 := ptr2.link;
                        if ptr1 = null then
                            buckets(n) := ptr3;
                        else
                            ptr1.link := ptr3;
                        end if;
                        Air_Traffic_Display.update_cws(threat => ptr2.target,
                                                       display_status => delete);
                        free_target(ptr2.target);
                        free_table_entry(ptr2);
                        ptr2 := ptr3;
                    else
                        ptr1 := ptr2;
                        ptr2 := ptr2.link;
                    end if;
                end loop;
            end loop;
        end loop;
exception
    when constraint_error => text_io.put_line("ptib CE");
    when numeric_error => text_io.put_line("ptib NE");
    when others => text_io.put_line("ptib Bozo error");
    end purge_target_information_blocks;


--
-- Tasks to process potential_threats in a specified
-- collision warning situation.
--
}
^stream!int(priority, (
^forall(c, cws, ( {
    task collision_warning_situation_{c.cws_name}
    is
        pragma Priority(6-{priority});
    end collision_warning_situation_{c.cws_name};
```

```
        task body collision_warning_situation_{c.cws_name}
        is
            msg : pt_handle;
        begin
            loop
--
-- Receive the next target and perform the desired processing
-- with it.
--
            Radar_Target_Priority_Buffer.receive_{c.cws_name}(msg);
}
            ^select(
                ^equal(c.alarm, {True}) -> ( {
            Audible_Alarm.ring_alarm({c.cws_name});
}
                )
            )
            ^select(
                ^equal(c.atc_msg, {True}) -> ( {
            Communication.send_atc_msg({c.cws_name});
}
                )
            )
            ^select(
                ^equal(c.inter_air_msg, {True}) -> ( {
            Communication.send_ia_msg({c.cws_name});
}
                )
            )
            ^select(
                ^equal(c.corrective, {True}) -> ( {
            Air_Traffic_Display.corrective_action_msg(threat => msg);
}
                )
            )
{           Air_Traffic_Display.update_cws(threat => msg,
                                          display_status => add_modify);
        end loop;
exception
    when constraint_error => text_io.put_line("cwss_{c.cws_name} CE");
    when numeric_error    => text_io.put_line("cwss_{c.cws_name} NE");
    when others           => text_io.put_line("cwss_{c.cws_name} Bozo error");
    end collision_warning_situation_{c.cws_name}; }
))
))
{
--
-- Task to obtain radar information
--
--
-- How long to wait before accessing the next radar data record. This
-- is only used for the simulation. When we have a real
```

```
-- radar device, the delay statement in the monitor_radar task body MUST be
-- removed.
--
    task get_radar_information is
        pragma Priority(9);
    end get_radar_information;

    task body get_radar_information is
        aircraft_id : string(1..8);
        sweep : integer;
        relative_bearing : Physical_Quantities.degrees;
        target_range : Physical_Quantities.nautical_mile;
        timestamp : Physical_Quantities.seconds;
        target : target_info(from => RADAR_SOURCE);
    begin
        delay Startup_Delay;
        loop
            Radar.get_radar_data(aircraft_id, sweep, relative_bearing,
target_range, timestamp);
--
-- Save the information and forward it to the processing task.
--
            target.aircraft_id := aircraft_id;
            target.target_range := target_range;
            target.relative_bearing := relative_bearing;
            target.timestamp := timestamp;
            target.sweep := sweep;
            Target_Buffer.send(target);
        end loop;
exception
    when constraint_error => text_io.put_line("gri CE");
    when numeric_error => text_io.put_line("gri NE");
    when others => text_io.put_line("gri Bozo error");
    end get_radar_information;


--
-- Task to obtain potential threat information from
-- the air traffic control device.
--
-- How long to wait before accessing the next atc data record. This is only
-- used during the simulation. When we have a real ATC device,
-- then the delay statement in the monitor_atc task body MUST
-- be removed.
--
--
    task get_atc_information is
        pragma Priority(9);
    end get_atc_information;

    task body get_atc_information is
        aircraft_id : string(1..8);
        altitude : Physical_Quantities.feet;
        airspeed : Physical_Quantities.knots;
```

```
            ground_track : Physical_Quantities.degrees;
            target_range : Physical_Quantities.nautical_mile;
            relative_bearing : Physical_Quantities.degrees;
            timestamp : Physical_Quantities.seconds;
            target : target_info(from => ATC_SOURCE);
      begin
            delay Startup_Delay;
            loop
                Air_Traffic_Control.get_atc_message(aircraft_id, altitude, airspeed,
                                                     ground_track, target_range,
relative_bearing,
                                                     timestamp);
--
-- Send the target information to the processing task.
--
               target.aircraft_id := aircraft_id;
               target.altitude := altitude;
               target.airspeed := airspeed;
               target.ground_track := ground_track;
               target.target_range := target_range;
               target.relative_bearing := relative_bearing;
               target.timestamp := timestamp;
               Target_Buffer.send(target);
            end loop;
      exception
         when constraint_error => text_io.put_line("gai CE");
         when numeric_error => text_io.put_line("gai NE");
         when others => text_io.put_line("gai Bozo error");
         end get_atc_information;


--
-- Task to process the potential threat target information received
-- from the radar and ATC devices.
--
      task update_potential_threat_information is
         pragma Priority(8);
      end update_potential_threat_information;

      task body update_potential_threat_information
      is
         target : pt_handle;
         info_block : target_info;
         new_status : cws_id;
      begin
         loop
--
-- Get next potential threat information block.
--
            Target_Buffer.receive(info_block);
--
-- Process the target information. If this is a new target,
-- then we must add it to the symbol table and set the appropriate
-- fields.
```

```
--
            target := lookup(info_block.aircraft_id);
          if target = null then
              target := install(info_block.aircraft_id);
            if info_block.from = RADAR_SOURCE then
                target.altitude_Y_valid := invalid;
                target.altitude_X := 0.0;
                target.altitude_Y := 0.0;
                target.time_Y := info_block.timestamp;
                target.time_X := info_block.timestamp;
                target.velocity_valid := invalid;
                target.climb_rate := 0.0;
                target.ground_track := 0.0;
                target.cws_status := normal;
                target.target_range := info_block.target_range;
                target.relative_bearing := info_block.relative_bearing;
                target.handle :=
Air_Traffic_Display_Device.null_display_handle;
                target.sweep := info_block.sweep;
            else
                target.altitude_Y := info_block.altitude;
                target.altitude_Y_valid := valid;
                target.time_Y := info_block.timestamp;
                target.altitude_X := info_block.altitude;
                target.time_X := info_block.timestamp;
                target.velocity := info_block.airspeed;
                target.velocity_valid := valid;
                target.climb_rate := 0.0;
                target.ground_track := info_block.ground_track;
                target.cws_status := normal;
                target.target_range := info_block.target_range;
                target.relative_bearing := info_block.relative_bearing;
                target.handle :=
Air_Traffic_Display_Device.null_display_handle;
                target.sweep := 0;
            end if;
          else
--
-- Update information for an existing target.
--
            if info_block.from = RADAR_SOURCE then
                target.sweep := info_block.sweep;
                target.target_range := info_block.target_range;
                target.relative_bearing := info_block.relative_bearing;
                target.time_X := target.time_Y;
                target.time_Y := info_block.timestamp;
            else
                if target.altitude_Y_valid = invalid then
                    target.altitude_X := info_block.altitude;
                else
                    target.altitude_X := target.altitude_Y;
                end if;
```

```
                        target.time_X := target.time_Y;
                        target.altitude_Y := info_block.altitude;
                        target.altitude_Y_valid := valid;
                        target.time_Y := info_block.timestamp;
                        target.velocity := info_block.airspeed;
                        target.velocity_valid := valid;
                        target.ground_track := info_block.ground_track;
                        target.target_range := info_block.target_range;
                        target.relative_bearing := info_block.relative_bearing;
                    end if;
    --
    -- Compute the climb rate for the target using the new information
    --
    begin
                target.climb_rate := Air_Craft_Motion.get_climb_rate(
                                        altitude_Y => target.altitude_Y,
                                        time_Y      => target.time_Y,
                                        altitude_X => target.altitude_X,
                                        time_X      => target.time_X);
    exception
       when constraint_error => text_io.put_line("upti -1- CE");
       when numeric_error => text_io.put_line("upti -1- NE");
       when others => text_io.put_line("upti -1- Bozo error");
    end;
           .   end if;
    --
    -- Determine the situation this target is in relative to the host
    -- aircraft and pass the target information along to the
    -- appropriate task for further processing.
    --
            new_status :=
    Collision_Warning_Situation_Status.determine_cws_status(target);
            if new_status /= target.cws_status and new_status /= normal then
               target.cws_status := new_status;
               Radar_Target_Priority_Buffer.send(target,
    cws_to_priority(target.cws_status));
            else
               Air_Traffic_Display.update_cws(threat => target,
                                              display_status => add_modify);
            end if;
         end loop;
    exception
       when constraint_error => text_io.put_line("upti CE");
       when numeric_error => text_io.put_line("upti NE");
       when others => text_io.put_line("upti Bozo error");
       end update_potential_threat_information;


    -------------------------------------------------------------------------

    -----


    --
    -- Returns whether the altitude value for the specified
    -- potential threat is valid.
```

```
--
   function altitude_valid(threat : in pt_handle) return boolean
   is
   begin
      return threat.altitude_Y_valid = valid;
   end altitude_valid;


--
-- Return the identification of the specified potential threat.
--
   function get_aircraft_id(threat : in pt_handle) return string
   is
   begin
      return threat.aircraft_id;
   end get_aircraft_id;


--
-- Return the current altitude of the specified potential threat. Exception
-- Altitude_Invalid is raised if the altitude valid is invalid.
--
   function get_altitude(threat : in pt_handle) return
Physical_Quantities.feet
   is
   begin
      return threat.altitude_Y;
   end get_altitude;


--
-- Returns the most recently measured climb rate for the
-- potential threat.
--
   function get_climb_rate(threat : in pt_handle) return
Physical_Quantities.fpm
   is
   begin
      return threat.climb_rate;
   end get_climb_rate;


--
-- Return the collision warning situation status of the specified potential
-- threat.
--
   function get_cws_status(threat : in pt_handle) return cws_id
   is
   begin
      return threat.cws_status;
   end get_cws_status;


--
-- Return the display handle for the specified potential threat.
--
   function get_display_handle(threat : in pt_handle)
                  return Air_Traffic_Display_Device.display_handle
   is
```

```
    begin
       return threat.handle;
    end get_display_handle;


    --
    -- Set the display handle for the specified potential threat.
    --
    procedure set_display_handle(threat : in pt_handle;
                                     handle : in
Air_Traffic_Display_Device.display_handle)
    is
    begin
       threat.handle := handle;
    end set_display_handle;


    --
    -- Return the ground track for the specified potential threat.
    --
    function get_ground_track(threat : in pt_handle) return
Physical_Quantities.degrees
    is
    begin
       return threat.ground_track;
    end get_ground_Track;


    --
    -- Return the potential threat partition.
    --
    function get_partition(threat : in pt_handle) return partition
    is
    begin
       return ID;
    end get_partition;


    --
    -- Returns the most recently measured range between the specified
    -- potential threat and the host aircraft.
    --
    function get_range(threat : in pt_handle) return
Physical_Quantities.nautical_mile
    is
    begin
       return threat.target_range;
    end get_range;


    --
    -- Returns the most recently measured relative_bearing of the
    -- specified potential threat.
    --
    function get_relative_bearing(threat . in pt_handle)
                                            return Physical_Quantities.degrees
    is
    begin
```

```
        return threat.relative_bearing;
    end get_relative_bearing;


  --
  -- Returns the most recently measured velocity of the specified
  -- potential threat. Exception Velocity_Invalid is raised when
  -- the velocity of the potential threat is invalid.
  --
    function get_velocity(threat : in pt_handle) return
Physical_Quantities.knots
    is
    begin
        return threat.velocity;
    end get_velocity;


  --
  -- Returns a status which indicates whether the velocity of the
  -- specified potential threat is valid.
  --
    function velocity_valid(threat : in pt_handle) return boolean
    is
    begin
        return threat.velocity_valid = valid;
    end velocity_valid;

end Potential_Threat;
}))}
```

16. Potential_Threat_Partition (PTP)

Spec

```
  --
  -- Potential_Threat_Partition (PTP)
  --
  -- This module determines which partition a potential_threat
  -- is a member of.
  --
  -- Either generic parameter altitude, airspeed, or both MUST be TRUE
  -- to have a legal instantiation.
  --
with Potential_Threat;
generic
    altitude : boolean;
    airspeed : boolean;

package Potential_Threat_Partition is

    function get_partition(threat : Potential_Threat.pt_handle)
                                    return Potential_Threat.partition;

end Potential_Threat_Partition;
```

Body

```
--
-- Potential_Threat_Partition (PTP)  package body
--
with Potential_Threat;
package body Potential_Threat_Partition is

    function get_partition(threat : Potential_Threat.pt_handle)
                                    return Potential_Threat.partition
    is
    begin
       if (altitude = TRUE and then airspeed = TRUE) then
          if Potential_Threat.altitude_valid(threat) and then
                        Potential_Threat.velocity_valid(threat) then
             return Potential_Threat.ID;
          else
             return Potential_Threat.UID;
          end if;
       elsif (altitude = TRUE) then
          if Potential_Threat.altitude_valid(threat) then
             return Potential_Threat.ID;
          else
             return Potential_Threat.UID;
          end if;
       elsif (airspeed = TRUE) then
          if Potential_Threat.velocity_valid(threat) then
             return Potential_Threat.ID;
          else
             return Potential_Threat.UID;
          end if;
       end if;
    end get_partition;

end Potential_Threat_Partition;
```

17. Radar (RADAR)

Spec

```
--
-- Radar (RADAR)  package spec
--
-- This module encapsulates the hardware / software interface to
-- the radar device. The primary hidden decisions are how to
-- obtain ran data for the aircraft_identification, sweep, relative
-- bearing, range, and timestamp; the scale and format
-- of these input data imtes; and the device-dependent operations
-- that must be applied to convert the raw data to the internal format
-- of the ATD/CWM system.
--
with Physical_Quantities;
package Radar is

    procedure get_radar_data(aircraft_id : out string;
                             sweep : out integer;
```

```
                                    relative_bearing : out
Physical_Quantities.degrees;
                                    target_range : out
Physical_Quantities.nautical_mile;
                                    timestamp : out Physical_Quantities.seconds);


end Radar;


--
-- Following package only for providing simulation data for
-- the Radar and ATC.
--
with Physical_Quantities;
package Simulation_Data is


--
-- Miscellaneous exceptions
--
   Out_Of_Host_Data : exception;
   Out_Of_Radar_Data : exception;


--
-- Procedure for providing information for a simulated
-- radar return.
--
   procedure get_sim_data(aircraft_id : out string;
                          sweep : out integer;
                          relative_bearing : out Physical_Quantities.degrees;
                          target_range : out
Physical_Quantities.nautical_mile);


--
-- Procedure for providing information for a simulated ATC input.
--
   procedure get_sim_data(aircraft_id : out string;
                          altitude : out Physical_Quantities.feet;
                          airspeed : out Physical_Quantities.knots;
                          ground_track : out Physical_Quantities.degrees;
                          target_range : out
Physical_Quantities.nautical_mile;
                          relative_bearing : out Physical_Quantities.degrees);


--
-- Procedure for providing information for a simulated navigation input.
--
   procedure get_sim_data(altitude : out Physical_Quantities.feet;
                          airspeed : out Physical_Quantities.knots;
                          ground_track : out Physical_Quantities.degrees;
                          latitude : out Physical_Quantities.latitude;
                          longitude : out Physical_Quantities.longitude);


end Simulation_Data;
```

Body

```
--
-- Radar (RADAR)   package body
--
-- This module encapsulates the hardware / software interface to
-- the radar device. The primary hidden decisions are how to
-- obtain ran data for the aircraft_identification, sweep, relative
-- bearing, range, and timestamp; the scale and format
-- of these input data imtes; and the device-dependent operations
-- that must be applied to convert the raw data to the internal format
-- of the ATD/CWM system.
--
with Physical_Quantities;
with Simulation_Data;
with Text_IO;
package body Radar is


    --
    --
    -- Get next radar return.
    --
    procedure get_radar_data(aircraft_id : out string;
                             sweep : out integer;
                             relative_bearing : out
Physical_Quantities.degrees;
                             target_range : out
Physical_Quantities.nautical_mile;
                             timestamp : out Physical_Quantities.seconds)
    is
    begin
        Simulation_Data.get_sim_data(aircraft_id, sweep, relative_bearing,
target_range);
        timestamp := Physical_Quantities.get_time;
exception
    when constraint_error => text_io.put_line("radar CE");
    when numeric_error =>    text_io.put_line("radar NE");
    when Simulation_Data.Out_Of_Radar_Data => text_io.put_line("radar OUT
RADAR");
    when others => text_io.put_line("radar BOZO");
    end get_radar_data;

end Radar;


--
-- Simulation_Data package
--
with Physical_Quantities;
with Radar;
with Air_Craft_Motion;
with Numerical_Algorithms;
with Text_IO;
package body Simulation_Data is

    package float_io is new text_io.float_io(float); use float_io;
```

```
--
-- Various Constants
--

   Tracked_Targets : constant := 5;

   Radar_Delay : constant := 2.0 / Tracked_Targets;
   ATC_Delay   : constant := 2.0 / Tracked_Targets;


--
-- Current radar sweep.
--

   sweep_counter : integer := 0;
   number_of_calls : integer := 1;


-------------------------------------------------------------------------
--
-- NAVIGATION DEVICE INPUT SIMULATION
--
-- Because we don't have a real navigation device, we are
-- going to use canned data. The data will describe a
-- hypothetical flight path for the host aircraft. To do this,
-- we will construct navigation data algorithmically
-- based on a given starting point. The algorithm
-- utilizes the following information to compute the navigation
-- records:
--
--     initial altitude
--     initial airspeed
--     initial ground_track
--     delta altitude
--     delta airspeed
--     delta ground_track
--
-- The algorithm computes an altitude, airspeed, and ground_track
-- per invocation. The 'delta' values represent the rate of change per
invocation
-- of the algorithm for the respective quantity.
--
-- Each starting point is considered the beginning of a different
-- scenario. We also need to specify how long to simulate a given scenario
-- in terms of the number of records (i.e., how many times to invoke
-- the algorithm).
--
-- The following Ada declarations are used to store
-- the information described above for each scenario.
--
   type scenario_info is
      record
         initial_altitude : Physical_Quantities.feet;
         initial_airspeed : Physical_Quantities.knots;
         initial_ground_track : Physical_Quantities.degrees;
         records : integer;
         delta_altitude : float;
```

```
                  delta_airspeed : float;
                  delta_ground_track : float;
               end record;

       scenario : array(1..16) of scenario_info := (
--
-- Scenario #1
--
       1 => (initial_altitude => 1000.0,
               initial_airspeed => 700.0,
               initial_ground_track => 0.0,
               records => 600,
               delta_altitude => 0.0,
               delta_airspeed => 0.0,
               delta_ground_track => 0.0),
--
-- Scenario #2
--
       2 => (initial_altitude => 1000.0,
               initial_airspeed => 700.0,
               initial_ground_track => 0.0,
               records => 300,
               delta_altitude => 10.0,
               delta_airspeed => 0.0,
               delta_ground_track => 0.0),
--
-- Scenario #3
--
       3 => (initial_altitude => 31000.0,
               initial_airspeed => 700.0,
               initial_ground_track => 0.0,
               records => 300,
               delta_altitude => -10.0,
               delta_airspeed => 0.0,
               delta_ground_track => 0.0),
--
-- Scenario #4
--
       4 => (initial_altitude => 1000.0,
               initial_airspeed => 500.0,
               initial_ground_track => 0.0,
               records => 150,
               delta_altitude => 0.0,
               delta_airspeed => 0.1,
               delta_ground_track => 0.0),

--
-- Scenario #5
--
       5 => (initial_altitude => 1000.0,
               initial_airspeed => 700.0,
               initial_ground_track => 0.0,
               records => 150,
```

```
                          delta_altitude => 0.0,
                          delta_airspeed => -0.1,
                          delta_ground_track => 0.0),
       --
       -- Scenario #6
       --
            6 => (initial_altitude => 1000.0,
                          initial_airspeed => 500.0,
                          initial_ground_track => 0.0,
                          records => 150,
                          delta_altitude => 10.0,
                          delta_airspeed => 0.1,
                          delta_ground_track => 0.0),
       --
       -- Scenario #7
       --
            7 => (initial_altitude => 31000.0,
                          initial_airspeed => 700.0,
                          initial_ground_track => 0.0,
                          records => 150,
                          delta_altitude => -10.0,
                          delta_airspeed => -0.1,
                          delta_ground_track => 0.0),
       --
       -- Scenario #8
       --
            8 => (initial_altitude => 1000.0,
                          initial_airspeed => 500.0,
                          initial_ground_track => 0.0,
                          records => 150,
                          delta_altitude => 10.0,
                          delta_airspeed => 0.1,
                          delta_ground_track => 0.0),
       --
       -- Scenario #9
       --
            9 => (initial_altitude => 31000.0,
                          initial_airspeed => 700.0,
                          initial_ground_track => 0.0,
                          records => 150,
                          delta_altitude => -10.0,
                          delta_airspeed => -0.1,
                          delta_ground_track => 0.0),
       --
       -- Scenario #10
       --
            10 => (initial_altitude => 1000.0,
                          initial_airspeed => 700.0,
                          initial_ground_track => 0.0,
                          records => 300,
                          delta_altitude => 0.0,
                          delta_airspeed => 0.0,
```

```
                    delta_ground_track => 0.1),
--
-- Scenario #11
--
     11 => (initial_altitude => 1000.0,
            initial_airspeed => 500.0,
            initial_ground_track => 0.0,
            records => 150,
            delta_altitude => 10.0,
            delta_airspeed => 0.1,
            delta_ground_track => 0.1),
--
-- Scenario #12
--
     12 => (initial_altitude => 31000.0,
            initial_airspeed => 700.0,
            initial_ground_track => 0.0,
            records => 150,
            delta_altitude => -10.0,
            delta_airspeed => -0.1,
            delta_ground_track => 0.1),
--
-- Scenario #13
--
     13 => (initial_altitude => 1000.0,
            initial_airspeed => 500.0,
            initial_ground_track => 0.0,
            records => 150,
            delta_altitude => 10.0,
            delta_airspeed => 0.1,
            delta_ground_track => 0.1),
--
-- Scenario #14
--
     14 => (initial_altitude => 31000.0,
            initial_airspeed => 700.0,
            initial_ground_track => 0.0,
            records => 150,
            delta_altitude => -10.0,
            delta_airspeed => -0.1,
            delta_ground_track => 0.1),
--
-- Scenario #15
--
     15 => (initial_altitude => 1000.0,
            initial_airspeed => 500.0,
            initial_ground_track => 0.0,
            records => 150,
            delta_altitude => 10.0,
            delta_airspeed => 0.1,
            delta_ground_track => 0.1),
--
```

```
-- Scenario #16
--
    16 => (initial_altitude => 31000.0,
           initial_airspeed => 700.0,
           initial_ground_track => 0.0,
           records => 150,
           delta_altitude => -10.0,
           delta_airspeed => -0.1,
           delta_ground_track => 0.1));

--
-- Pointers to current host aircraft navigation data.
--
   host_record_count : integer := 1;
   host_data_index : integer := scenario'first;

   host_current_altitude : Physical_Quantities.feet :=

scenario(host_data_index).initial_altitude;
   host_current_airspeed : Physical_Quantities.knots :=

scenario(host_data_index).initial_airspeed;
   host_current_ground_track : Physical_Quantities.degrees :=

scenario(host_data_index).initial_ground_track;

   host_climb_rate : Physical_Quantities.fpm := 0.0;

--
-- Previous host aircraft information so that we can
-- compute climb_rate for the host.
--
   previous_host_altitude : Physical_Quantities.feet := 0.0;
   previous_host_altitude_time : Physical_Quantities.seconds;

--------------------------------------------------------------------
-----
--
-- RADAR and ATC INPUT SIMULATION
--
--
-- Each target has a scenario specified by an initial starting
-- condition. From this, we predict the new range and relative
-- bearing as well as updating the altitude. We are
-- assuming constant airspeed and ground_track for the targets.
--
-- The target scenarios are defined by the following record.
--
   type target_scenario is
      record
          id : string(1..8);
          initial_altitude : Physical_Quantities.feet;
          initial_airspeed : Physical_Quantities.knots;
          initial_ground_track : Physical_Quantities.degrees;
```

```
                initial_target_range : Physical_Quantities.nautical_mile;
                initial_relative_bearing : Physical_Quantities.degrees;
                initial_climb_rate : Physical_Quantities.fpm;
            end record;


    --
    -- Target scenarios
    --
        targets : array(1..12) of target_scenario := (
            1 => (id => "PT_00034",
                    initial_altitude => 1000.0,
                    initial_airspeed => 700.0,
                    initial_ground_track => 180.0,
                    initial_target_range => 60.0,
                    initial_relative_bearing => 0.0,
                    initial_climb_rate => 0.0),
            2 => (id => "PT_06789",
                    initial_altitude => 1000.0,
                    initial_airspeed => 700.0,
                    initial_ground_track => 30.0,
                    initial_target_range => 60.0,
                    initial_relative_bearing => 225.0,
                    initial_climb_rate => 0.0),
            3 => (id => "BlueBomb",
                    initial_altitude => 1000.0,
                    initial_airspeed => 700.0,
                    initial_ground_track => 210.0,
                    initial_target_range => 60.0,
                    initial_relative_bearing => 45.0,
                    initial_climb_rate => 0.0),
            4 => (id => "Stealth ",
                    initial_altitude => 1000.0,
                    initial_airspeed => 700.0,
                    initial_ground_track => 90.0,
                    initial_target_range => 60.0,
                    initial_relative_bearing => 300.0,
                    initial_climb_rate => 0.0),
            5 => (id => "SynThsis",
                    initial_altitude => 1000.0,
                    initial_airspeed => 700.0,
                    initial_ground_track => 270.0,
                    initial_target_range => 60.0,
                    initial_relative_bearing => 75.0,
                    initial_climb_rate => 0.0),
            6 => (id => "Snoopy  ",
                    initial_altitude => 5000.0,
                    initial_airspeed => 700.0,
                    initial_ground_track => 70.0,
                    initial_target_range => 45.0,
                    initial_relative_bearing => 325.0,
                    initial_climb_rate => 0.0),
            7 => (id => "F-111   ",
                    initial_altitude => 3000.0,
```

```
                         initial_airspeed => 700.0,
                         initial_ground_track => 180.0,
                         initial_target_range => 60.0,
                         initial_relative_bearing => 340.0,
                         initial_climb_rate => 0.0),
          8 => (id => "UA-12345",
                         initial_altitude => 1500.0,
                         initial_airspeed => 700.0,
                         initial_ground_track => 135.0,
                         initial_target_range => 50.0,
                         initial_relative_bearing => 270.0,
                         initial_climb_rate => 0.0),
          9 => (id => "ZeePlane",
                         initial_altitude => 5000.0,
                         initial_airspeed => 700.0,
                         initial_ground_track => 190.0,
                         initial_target_range => 60.0,
                         initial_relative_bearing => 0.0,
                         initial_climb_rate => 0.0),
         10 => (id => "Mt.Reuse",
                         initial_altitude => 3500.0,
                         initial_airspeed => 700.0,
                         initial_ground_track => 270.0,
                         initial_target_range => 60.0.
                         initial_relative_bearing => 45.0,
                         initial_climb_rate => 0.0),
         11 => (id => "UFO #001",
                         initial_altitude => 25000.0,
                         initial_airspeed => 700.0.
                         initial_ground_track => 90.0,
                         initial_target_range => 55.0.
                         initial_relative_bearing => 315.0,
                         initial_climb_rate => 0.0),
         12 => (id => "Blimp 99",
                         initial_altitude => 24000.0,
                         initial_airspeed => 700.0,
                         initial_ground_track => 80.0,
                         initial_target_range => 60.0,
                         initial_relative_bearing => 350.0,
                         initial_climb_rate => 0.0)
     );


   --
   -- We only track a certain number of targets at any one time. The following
   -- record is used to keep track of a target while it is being tracked.
   --
      type tracked_target is
        record
           tracked : boolean;
           id : string(1..8);
           altitude : Physical_Quantities.feet;
           airspeed : Physical_Quantities.knots;
           ground_track : Physical_Quantities.degrees;
```

```
            target_range : Physical_Quantities.nautical_mile;
            relative_bearing : Physical_Quantities.degrees;
            climb_rate : Physical_Quantities.fpm;
            timestamp : Physical_Quantities.seconds;
        end record;


    pt : array(1..Tracked_Targets) of tracked_target := (
       1 => (tracked => false,
             id => "          ",
             altitude => 0.0,
             airspeed => 0.0,
             ground_track => 0.0,
             target_range => 0.0,
             relative_bearing => 0.0,
             climb_rate => 0.0,
             timestamp => 0.0),
       2 => (tracked => false.
             id => "          ",
             altitude => 0.0,
             airspeed => 0.0,
             ground_track => 0.0,
             target_range => 0.0,
             relative_bearing => 0.0,
             climb_rate => 0.0,
             timestamp => 0.0),
       3 => (tracked => false,
             id => "          ",
             altitude => 0.0,
             airspeed => 0.0,
             ground_track => 0.0,
             target_range => 0.0,
             relative_bearing => 0.0,
             climb_rate => 0.0,
             timestamp => 0.0),
       4 => (tracked => false,
             id => "          ",
             altitude => 0.0,
             airspeed => 0.0,
             ground_track => 0.0,
             target_range => 0.0,
             relative_bearing => 0.0,
             climb_rate => 0.0,
             timestamp => 0.0),
       5 => (tracked => false,
             id => "          ",
             altitude => 0.0,
             airspeed => 0.0,
             ground_track => 0.0,
             target_range => 0.0,
             relative_bearing => 0.0,
             climb_rate => 0.0,
```

```
                    timestamp => 0.0)
    );


    --
    -- Radar target index. Indicates what target to fetch next
    -- when get_sim_data is invoked.
    --

    target_index : integer := 1;
    next_target : integer := 1;


    --
    -- ATC target index. Indicates what target to fetch next
    -- when get_sim_data is invoked for an ATC input.
    --

    atc_target_index : integer := 1;


    --------------------------------------------------------------------------
    -- Simulated navigation device input.
    --
    -- Read the host aircraft navigation data ad return the converted
    -- information to the calling program.
    --

    procedure get_sim_data(altitude : out Physical_Quantities.feet;
                           airspeed : out Physical_Quantities.knots;
                           ground_track : out Physical_Quantities.degrees;
                           latitude : out Physical_Quantities.latitude;
                           longitude : out Physical_Quantities.longitude)
    is
        current_time : Physical_Quantities.seconds;
    begin
    --
    -- If already at the end of the simulation data, then punt with the
    -- appropriate exception. Otherwise, generate the next navigation
    -- data record for the current scenario.
    --

        if host_record_count > scenario(host_data_index).records then
            host_data_index := host_data_index + 1;
            if host_data_index > scenario'last then
                raise Out_Of_Host_Data;
            end if;
            host_current_altitude := scenario(host_data_index).initial_altitude;
            host_current_airspeed := scenario(host_data_index).initial_airspeed;
            host_current_ground_track :=
scenario(host_data_index).initial_ground_track;
            host_record_count := 1;
        end if;
    --
    -- Compute climb rate before we assign the current values of altitude,
    -- airspeed, and ground_track to the 'out' parameters.
    --

        current_time := Physical_Quantities.get_time;
        if previous_host_altitude /= 0.0 then
            host_climb_rate := Air_Craft_Motion.get_climb_rate(
```

```
                                            host_current_altitude,
                                            current_time,
                                            previous_host_altitude,
                                            previous_host_altitude_time);
        end if;
        previous_host_altitude := host_current_altitude;
        previous_host_altitude_time := current_time;
   --
   -- Now that we have the climb rate, we can update the 'out' parameters
   -- and adjust the altitude, airspeed, and ground_track values
   -- for the next time this routine is called.
   --
        altitude := host_current_altitude;
        airspeed := host_current_airspeed;
        ground_track := host_current_ground_track;
        latitude := 0.0;
        longitude := 0.0;
        host_current_altitude := host_current_altitude +

scenario(host_data_index).delta_altitude;
        host_current_airspeed := host_current_airspeed +

scenario(host_data_index).delta_airspeed;
        host_current_ground_track := host_current_ground_track +

scenario(host_data_index).delta_ground_track;
        host_record_count := host_record_count + 1;
      end get_sim_data;

----------------------------------------------------------------------
-- Simulated Radar and ATC input
--
-- The current altitude, airspeed, ground_track, relative_bearing,
-- and range is updated only when we need to provide new
-- information for the radar. A target is considered "tracked"
-- as long as its range is within the surveillance area. After
-- that, we exclude that target and begin tracking a "new"
-- target (i.e., that is, we start a new scenario).
--
   procedure get_sim_data(aircraft_id : out string;
                          sweep : out integer;
                          relative_bearing : out Physical_Quantities.degrees;
                          target_range : out
Physical_Quantities.nautical_mile)
   is
        range_xy : Physical_Quantities.nautical_mile;
        pt_velocity_xy : Physical_Quantities.knots;    -- X-Y velocity of
potential_threat
        ha_velocity_xy : Physical_Quantities.knots;    -- X-Y velocity of
host_aircraft
        current_time : Physical_Quantities.seconds;
        elapsed_time : Physical_Quantities.seconds;
        new_range : Physical_Quantities.nautical_mile;
```

```ada
        xh, yh, zh : float;
        xpt, ypt, zpt : float;
        temp_1, temp_2, temp_3 : float;
        new_relative_bearing : Physical_Quantities.degrees;
    begin
        delay Radar_Delay;
        current_time := Physical_Quantities.get_time;
--
-- Reference target indicated by target_index. If this slot indicates
-- that we are not tracking a target, then get the next
-- target. If none exist, then raise Out_Of_Radar_Data and punt.
--
        if pt(target_index).tracked = false then
            if next_target > targets'last then
                raise Out_Of_Radar_Data;
            end if;
            pt(target_index).id := targets(next_target).id;
            pt(target_index).altitude := targets(next_target).initial_altitude;
            pt(target_index).airspeed := targets(next_target).initial_airspeed;
            pt(target_index).ground_track :=
targets(next_target).initial_ground_track;
            pt(target_index).target_range :=
targets(next_target).initial_target_range;
            pt(target_index).relative_bearing :=
targets(next_target).initial_relative_bearing;
            pt(target_index).climb_rate :=
targets(next_target).initial_climb_rate;
            pt(target_index).tracked := true;
            next_target := next_target + 1;
        else
--
-- Compute new range for this target
--
begin
            range_xy := Air_Craft_Motion.get_range_xy(
                            pt(target_index).target_range,
                            pt(target_index).altitude,
                            host_current_altitude);

            pt_velocity_xy := Air_Craft_Motion.get_velocity_xy(
                            pt(target_index).airspeed,
                            pt(target_index).climb_rate);

            ha_velocity_xy :=
Air_Craft_Motion.get_velocity_xy(host_current_airspeed,

host_climb_rate);
exception
    when constraint_error =>
        text_io.put_line("get sim data radar - 42 CE");
    when numeric_error =>
        text_io.put_line("get sim data radar - 42 NE");
    when others =>
```

```
            text_io.put_line("get sim data radar - 42 Bozo error");
end;


--
-- Elapsed time is the difference between the value of the timestamp
-- in the threat(target_index) and the current time.
--
        elapsed_time := current_time - pt(target_index).timestamp;
        pt(target_index).timestamp := current_time;
--
-- First, compute the new location of the host aircraft.
--
begin
        xh := ha_velocity_xy *
Numerical_Algorithms.cos(host_current_ground_track) *
                    (float(elapsed_time) / 3600.0);
        yh := ha_velocity_xy *
Numerical_Algorithms.sin(host_current_ground_track) *
                    (float(elapsed_time) / 3600.0);
        zh := host_climb_rate * (float(elapsed_time) / 60.0);
exception
    when constraint_error =>
        text_io.put_line("get sim data radar - 43 CE");
    when numeric_error =>
        text_io.put_line("get sim data radar - 43 NE");
    when others =>
        text_io.put_line("get sim data radar - 43 Bozo error");
end;
--
-- Next, compute the new location of the threat.
--
begin
        xpt := range_xy *
Numerical_Algorithms.cos(pt(target_index).relative_bearing) +
                    pt_velocity_xy *
Numerical_Algorithms.cos(pt(target_index).ground_track) *
                        (float(elapsed_time) / 3600.0);
        ypt := range_xy *
Numerical_Algorithms.sin(pt(target_index).relative_bearing) +
                    pt_velocity_xy *
Numerical_Algorithms.sin(pt(target_index).ground_track) *
                        (float(elapsed_time) / 3600.0);
        zpt := (pt(target_index).altitude - host_current_altitude) +
                        (pt(target_index).climb_rate * (float(elapsed_time) /
60.0) - zh);
exception
    when constraint_error =>
        text_io.put_line("get sim data radar - 44 CE");
    when numeric_error =>
        text_io.put_line("get sim data radar - 44 NE");
    when others =>
        text_io.put_line("get sim data radar - 44 Bozo error");
end;
```

```
--
-- We can now compute the distance is then computed using
--
--        range = ((xpt-xh)**2 + (ypt-yh)**2 + (zpt-zh)**2) ** 0.5
--
begin
        temp_1 := xpt - xh;
        temp_2 := ypt - yh;
        temp_3 := (zpt - zh) / Physical_Quantities.nautical_mile_to_feet;
        new_range := Numerical_Algorithms.sqrt(temp_1 * temp_1 + temp_2 *
temp_2 +
                                                                    temp_3 *
temp_3);
exception
   when constraint_error =>
      text_io.put_line("get sim data radar - 45 CE");
   when numeric_error =>
      text_io.put_line("get sim data radar - 45 NE");
   when others =>
      text_io.put_line("get sim data radar - 45 Bozo error");
end;
--
-- Boundary check. If the predicted range is outside the surveillance area,
-- then move on to the next target. If no more targets exist, then raise
-- Out_Of_Radar_Data exception and punt.
--
        if new_range > 61.0 then
            if next_target > targets'last then
               raise Out_Of_Radar_Data;
            end if;
            pt(target_index).tracked := true;
            pt(target_index).id := targets(next_target).id;
            pt(target_index).altitude :=
targets(next_target).initial_altitude;
            pt(target_index).airspeed :=
targets(next_target).initial_airspeed;
            pt(target_index).ground_track :=
targets(next_target).initial_ground_track;
            pt(target_index).target_range :=
targets(next_target).initial_target_range;
            pt(target_index).relative_bearing :=
targets(next_target).initial_relative_bearing;
            pt(target_index).climb_rate :=
targets(next_target).initial_climb_rate;
            next_target := next_target + 1;
        else
--
-- Store the new range and altitude. Next, we compute the
-- predicted relative bearing for this target.
--
begin
            pt(target_index).target_range := new_range;
```

```
               pt(target_index).altitude := pt(target_index).altitude +
                              pt(target_index).climb_rate *
(float(elapsed_time) / 60.0);
exception
   when constraint_error =>
      text_io.put_line("get sim data radar - 46 CE");
   when numeric_error =>
      text_io.put_line("get sim data radar - 46 NE");
   when others =>
      text_io.put_line("get sim data radar - 46 Bozo error");
end;
begin
          range_xy := Air_Craft_Motion.get_range_xy(new_range,

pt(target_index).altitude,

host_current_altitude);
exception
   when constraint_error =>
      text_io.put_line("get sim data radar - 47 CE");
   when numeric_error =>
      text_io.put_line("get sim data radar - 47 NE");
   when others =>
      text_io.put_line("get sim data radar - 47 Bozo error");
end;
begin
          new_relative_bearing := Physical_Quantities.degrees(
                                Numerical_Algorithms.arccos(
                                   float((xpt - xh) / range_xy)) *

Physical_Quantities.radian_to_degree);
exception
   when constraint_error =>
      text_io.put_line("get sim data radar - 48 CE");
   when numeric_error =>
      text_io.put_line("get sim data radar - 48 NE");
   when others =>
      text_io.put_line("get sim data radar - 48 Bozo error");
      text_io.put("ID: "); text_io.put_line(pt(target_index).id);
      text_io.put("Alt: ");   put(float(pt(target_index).altitude), aft=>1,
exp=>0); text_io.new_line;
      text_io.put("Vel: ");   put(float(pt(target_index).airspeed), aft=>1,
exp=>0); text_io.new_line;
      text_io.put("GT: ");    put(float(pt(target_index).ground_track), aft=>1,
exp=>0); text_io.new_line;
      text_io.put("TR: ");    put(float(pt(target_index).target_range),aft=>1,
exp=>0); text_io.new_line;
      text_io.put("RB: ");
put(float(pt(target_index).relative_bearing),aft=>1,exp=>0); text_io.new_line;
      text_io.put("CR: ");
put(float(pt(target_index).climb_rate),aft=>1,exp=>0); text_io.new_line;
      text_io.put("xh: "); put(float(xh)); text_io.new_line;
```

```
      text_io.put("yh: "); put(float(yh)); text_io.new_line;
      text_io.put("zh: "); put(float(zh)); text_io.new_line;
      text_io.put("xpt: "); put(float(xpt)); text_io.new_line;
      text_io.put("ypt: "); put(float(ypt)); text_io.new_line;
      text_io.put("zpt: "); put(float(zpt)); text_io.new_line;
      text_io.put("range_xy: "); put(range_xy, aft=>2, exp=>0);
text_io.new_line;
      text_io.put("temp_1: "); put(temp_1, aft=>2, exp=>0); text_io.new_line;
      text_io.put("temp_2: "); put(temp_2, aft=>2, exp=>0); text_io.new_line;
      text_io.put("temp_3: "); put(temp_3, aft=>2, exp=>0); text_io.new_line;
      text_io.put("HA alt: "); put(float(host_current_altitude), aft=>2,
exp=>0); text_io.new_line;
end;
            if ypt < 0.0 then
               new_relative_bearing := 360.0 - new_relative_bearing;
            end if;
            pt(target_index).relative_bearing := new_relative_bearing;
         end if;
      end if;
--
-- Assign the aircraft_id, relative_bearing, and range to the
-- 'out' parameters before returning.
--
begin
      aircraft_id := pt(target_index).id;
      relative_bearing := pt(target_index).relative_bearing;
      target_range := pt(target_index).target_range;
      if number_of_calls = Tracked_Targets then
         sweep_counter := sweep_counter + 1;
         number_of_calls := 1;
      else
         number_of_calls := number_of_calls + 1;
      end if;
      sweep := sweep_counter;
exception
   when constraint_error =>
      text_io.put_line("get sim data radar - 49 CE");
   when numeric_error =>
      text_io.put_line("get sim data radar - 49 NE");
   when others =>
      text_io.put_line("get sim data radar - 49 Bozo error");
end;
--
-- Update fields for this target before returning.
--
      pt(target_index).timestamp := current_time;
--
-- Update target_index so that it will reference the next target
-- when get_sim_data is called again.
--
      target_index := target_index mod Tracked_Targets + 1;
exception
```

```
    when constraint_error =>
        text_io.put_line("get_sim_data radar CE");
        text_io.put_line("Target information");
        text_io.put("ID: "); text_io.put_line(pt(target_index).id);
        text_io.put("Alt: ");    put(float(pt(target_index).altitude), aft=>1,
exp=>0); text_io.new_line;
        text_io.put("Vel: ");    put(float(pt(target_index).airspeed), aft=>1,
exp=>0); text_io.new_line;
        text_io.put("GT: ");     put(float(pt(target_index).ground_track), aft=>1,
exp=>0); text_io.new_line;
        text_io.put("TR: ");     put(float(pt(target_index).target_range),aft=>1,
exp=>0); text_io.new_line;
        text_io.put("RB: ");
put(float(pt(target_index).relative_bearing),aft=>1,exp=>0); text_io.new_line;
        text_io.put("CR: ");
put(float(pt(target_index).climb_rate),aft=>1,exp=>0); text_io.new_line;

        text_io.put("range xy: "); put(float(range_xy)); text_io.new_line;
        text_io.put("pt_velocity_xy: "); put(float(pt_velocity_xy));
text_io.new_line;
        text_io.put("ha_velocity_xy: "); put(float(ha_velocity_xy));
text_io.new_line;
        text_io.put("xh: "); put(float(xh)); text_io.new_line;
        text_io.put("yh: "); put(float(yh)); text_io.new_line;
        text_io.put("zh: "); put(float(zh)); text_io.new_line;
        text_io.put("xpt: "); put(float(xpt)); text_io.new_line;
        text_io.put("ypt: "); put(float(ypt)); text_io.new_line;
        text_io.put("zpt: "); put(float(zpt)); text_io.new_line;

    when numeric_error =>    text_io.put_line("get_sim_data radar NE");
    when Out_Of_Radar_Data => text_io.put_line("get_sim_data radar OUT RADAR");
    when Numerical_Algorithms.root_negative =>
        text_io.put_line("get_sim_data radar SQRT negative");
        text_io.put("temp_1: "); put(temp_1, aft=>2, exp=>0);
        text_io.put("temp_2: "); put(temp_2, aft=>2, exp=>0);
        text_io.put("temp_3: "); put(temp_3, aft=>2, exp=>0);
    when others =>
        text_io.put_line("get_sim_data radar BOZO");
        text_io.put_line("Host data index: " & integer'image(host_data_index));
        text_io.put("range xy: "); put(float(range_xy), aft=>1, exp=>0);
text_io.new_line;
        text_io.put("temp_1: "); put(temp_1, aft=>2, exp=>0); text_io.new_line;
        text_io.put("temp_2: "); put(temp_2, aft=>2, exp=>0); text_io.new_line;
        text_io.put("temp_3: "); put(temp_3, aft=>2, exp=>0); text_io.new_line;
    end get_sim_data;


--
-- Return current information to simulate an ATC input.
--
    procedure get_sim_data(aircraft_id : out string;
                           altitude : out Physical_Quantities.feet;
                           airspeed : out Physical_Quantities.knots;
                           ground_track : out Physical_Quantities.degrees;
```

```
                                  target_range : out
Physical_Quantities.nautical_mile:
                                  relative_bearing : out Physical_Quantities.degrees)
    is
    begin
       loop
          delay ATC_Delay;
          if pt(atc_target_index).tracked = true then
             aircraft_id := pt(atc_target_index).id;
             altitude := pt(atc_target_index).altitude;
             airspeed := pt(atc_target_index).airspeed;
             ground_track := pt(atc_target_index).ground_track;
             target_range := pt(atc_target_index).target_range;
             relative_bearing := pt(atc_target_index).relative_bearing;
             atc_target_index := atc_target_index mod Tracked_Targets + 1;
             return;
          end if;
       end loop;
    end get_sim_data;

    end Simulation_Data;
```

18. Situation_Dynamics (SD)

Spec

```
--
-- Situation_Dynamics (SD) package spec
--
-- The hidden decisions of this module are how physical models can
-- be put together to predict a future situation starting from
-- a known state history.
--
with Physical_Quantities;
with Potential_Threat;
package Situation_Dynamics is


--
-- Returns the predicted elapsed time before the host_aircraft
-- and specified potential_threat reach their closest range.
--
    function get_elapsed_time(threat : in Potential_Threat.pt_handle)
                                           return Physical_Quantities.seconds;
--
-- Returns the predicted closest range between the host_aircraft
-- and specified potential_threat assuming constant velocity,
-- climb rate, and ground track for both aircraft.
--
    function get_msd(threat : in Potential_Threat.pt_handle)
                                           return Physical_Quantities.feet;


end Situation_Dynamics;
```

Body

```
--
-- Situation_Dynamics (SD) package body
--
-- The hidden decisions of this module are how physical models can
-- be put together to predict a future situation starting from
-- a known state history.
--
with Physical_Quantities;
with Air_Craft_Motion;
with Potential_Threat;
with Host_Aircraft;
with Numerical_Algorithms;
with Text_IO;
package body Situation_Dynamics is

    package float_io is new text_io.float_io(float); use float_io;


    --
    -- The time_to_intersect (elapsed time) computation assumes
    -- constant velocity, ground_track, and climb_rate for both
    -- the potential threat and host aircraft.
    --
    -- The range between the host aircraft (ha) and a potential threat (pt)
    -- at a given point in time is a function of their respective
    -- locations in space.
    --
    --             ------------------------------------------------------
    --          \   /
    --   range = \  / (pt_x - ha_x)**2 + (pt_y - ha_y)**2 + (pt_z - ha_z)**2
    --            \/
    --
    -- The location of an aircraft over time (assuming constant velocity,
    -- ground track, and climb_rate) is
    --
    --   x = x0 + velocity_xy * cos(ground_track) * time
    --   y = y0 + velocity_xy * sin(ground_track) * time
    --   z = z0 + climb_rate * time
    --
    -- Quantity (x0, y0, z0) denote the aircraft's initial location
    -- in space. "cos" and "sin" are the trigonometric functions
    -- sine and cosine, and velocity_xy is the horizontal component
    -- of the velocity (i.e., the velocity component that lies in
    -- the X-Y plane).
    --
    -- The potential threat location is always relative to the
    -- host aircraft. By assuming that the origin of the
    -- rectangular coordinate system is given by the host aircraft
    -- location, an initial position of the potential threat
    -- relative to the host aircraft is given by:
    --
    --     pt_x0 = range_xy * cos(pt_relative_bearing)
```

```
--      pt_y0 = range_xy * sin(pt_relative_bearing)
--      pt_z0 = pt_altitude - ha_altitude
--
-- range_xy is the range component that lies in the X-Y plane. Furthermore,
-- pt_altitude is the altitude of the potential threat (ha_altitude
-- is the altitude of the host aircraft).
--
-- Using these equations, the range between the host aircraft and
-- potential threat can be expressed as a function of time.
-- Taking the first derivative, setting it equal to zero, and
-- solving for time yields the time_to_intersect (i.e., how much
-- time elapses).
--
   function get_elapsed_time(threat : in Potential_Threat.pt_handle)
                                      return Physical_Quantities.seconds
   is
      range_xy : Physical_Quantities.nautical_mile;   -- X-Y range component
      pt_velocity_xy : Physical_Quantities.knots;     -- X-Y velocity of
potential_threat
      ha_velocity_xy : Physical_Quantities.knots;     -- X-Y velocity of
host_aircraft
         temp_1, temp_2, temp_3, temp_4 : float;
      begin
begin
      range_xy := Air_Craft_Motion.get_range_xy(
                             Potential_Threat.get_range(threat),
                             Potential_Threat.get_altitude(threat),
                             Host_Aircraft.get_altitude);
exception
   when constraint_Error =>
      text_io.put_line("sd 1 - CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error     =>
      text_io.put_line("sd 1 - NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others            =>
      text_io.put_line("sd 1 - Bozo error");
end ;
begin
      pt_velocity_xy := Air_Craft_Motion.get_velocity_xy(
                             Potential_Threat.get_velocity(threat),
                             Potential_Threat.get_climb_rate(threat));

      ha_velocity_xy :=
Air_Craft_Motion.get_velocity_xy(Host_Aircraft.get_velocity,

Host_Aircraft.get_climb_rate);
exception
   when constraint_Error =>
      text_io.put_line("sd 2 - CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error     =>
      text_io.put_line("sd 2 - NE ** " &
```

```
Potential_Threat.get_aircraft_id(threat));
   when others            => text_io.put_line("sd 2 - Bozo error");
end ;
begin
      temp_1 := pt_velocity_xy *
            Numerical_Algorithms.cos(Potential_Threat.get_ground_track(threat))
-
                  ha_velocity_xy *

Numerical_Algorithms.cos(Host_Aircraft.get_ground_track);

exception
   when constraint_Error =>
      text_io.put_line("sd 3 - CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error    =>
      text_io.put_line("sd 3 - NE ** " &
Potential_Threat.get_aircraft_id(threat)):
   when others            => text_io.put_line("sd 3 - Bozo error");
end ;
begin
      temp_2 := pt_velocity_xy *
            Numerical_Algorithms.sin(Potential_Threat.get_ground_track(threat))
-
                  ha_velocity_xy *

Numerical_Algorithms.sin(Host_Aircraft.get_ground_track);

exception
   when constraint_Error =>
      text_io.put_line("sd 4 - CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error    =>
      text_io.put_line("sd 4 - NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others            => text_io.put_line("sd 4 - Bozo error");
end ;
begin
      temp_3 := (Potential_Threat.get_climb_rate(threat) -
Host_Aircraft.get_climb_rate) /
                  Physical_Quantities.knot_to_fpm;

exception
   when constraint_Error =>
      text_io.put_line("sd 5 - CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error    =>
      text_io.put_line("sd 5 - NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others            => text_io.put_line("sd 5 - Bozo error");
end ;
begin
      temp_4 := -((
```

```
            range_xy *

Numerical_Algorithms.cos(Potential_Threat.get_relative_bearing(threat)) *
               temp_1 +
            range_xy *

  .

Numerical_Algorithms.sin(Potential_Threat.get_relative_bearing(threat)) *
               temp_2 +
            (Potential_Threat.get_altitude(threat) -
                    Host_Aircraft.get_altitude) * temp_3) /
               (temp_1*temp_1 + temp_2*temp_2 + temp_3*temp_3)) * 3600.0;
exception
   when constraint_Error =>
       text_io.put_line("sd 6 - CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error    =>
       text_io.put_line("sd 6 - NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others              => text_io.put_line("sd 6 - Bozo error");
end ;


--
-- Since time must be positive, adjust result accordingly.
--
begin
      if temp_4 < 0.0 then
         return Physical_Quantities.seconds(-temp_4);
      else
         return Physical_Quantities.seconds(temp_4);
      end if:
exception
   when constraint_Error =>
       text_io.put_line("sd 7 - CE ** " &
Potential_Threat.get_aircraft_id(threat);
   when numeric_error    =>
       text_io.put_line("sd 7 - NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others              => text_io.put_line("sd 7 - Bozo error");
end ;
   end get_elapsed_time;


--
-- Determine the minimal separation distance for a specified potential threat
-- and the host aircraft assuming constant velocity, ground_track, and
-- climb_rate for each.
--
-- This is computed by first determining how much
-- time elapses before the specified potential threat and host aircraft
-- are closest to each other. From this, we predict their locations
-- in 3-dimensional space assuming constant velocity, ground_track, and
-- climb_rate for each. From their respective predicted locations, we
-- can then compute the range between each other.
--
```

```
    function get_msd(threat : in Potential_Threat.pt_handle)
                                        return Physical_Quantities.feet
    is
        range_xy : Physical_Quantities.nautical_mile;   -- X-Y range component
        pt_velocity_xy : Physical_Quantities.knots;     -- X-Y velocity of
potential_threat
        ha_velocity_xy : Physical_Quantities.knots;     -- X-Y velocity of
host_aircraft
        time : float;
        temp_1 : float;
        temp_2 : float;
        temp_3 : float;
    begin
begin
        time := float(get_elapsed_time(threat));
exception
    when constraint_error =>
        text_io.put_line("sd - 21 CE ** " &
Potential_Threat.get_aircraft_id(threat));
    when numeric_error =>
        text_io.put_line("sd - 21 NE ** " &
Potential_Threat.get_aircraft_id(threat));
    when others =>
        text_io.put_line("sd - 21 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
begin
        range_xy := Air_Craft_Motion.get_range_xy(
                            Potential_Threat.get_range(threat),
                            Potential_Threat.get_altitude(threat),
                            Host_Aircraft.get_altitude);

exception
    when constraint_error =>
        text_io.put_line("sd - 22 CE ** " &
Potential_Threat.get_aircraft_id(threat));
    when numeric_error =>
        text_io.put_line("sd - 22 NE ** " &
Potential_Threat.get_aircraft_id(threat));
    when others =>
        text_io.put_line("sd - 22 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
begin
        pt_velocity_xy := Air_Craft_Motion.get_velocity_xy(
                            Potential_Threat.get_velocity(threat),
                            Potential_Threat.get_climb_rate(threat));

exception
    when constraint_error =>
        text_io.put_line("sd - 23 CE ** " &
Potential_Threat.get_aircraft_id(threat));
    when numeric_error =>
```

```
        text_io.put_line("sd - 23 NE ** " &
Potential_Threat.get_aircraft_id(threat));
    when others =>
        text_io.put_line("sd - 23 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
begin
        ha_velocity_xy :=
Air_Craft_Motion.get_velocity_xy(Host_Aircraft.get_velocity,

Host_Aircraft.get_climb_rate);
exception
    when constraint_error =>
        text_io.put_line("sd - 24 CE ** " &
Potential_Threat.get_aircraft_id(threat));
    when numeric_error =>
        text_io.put_line("sd - 24 NE ** " &
Potential_Threat.get_aircraft_id(threat));
    when others =>
        text_io.put_line("sd - 24 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
--
-- temp_1 holds the relative difference between the
-- host aircraft and potential threat at the elapsed time in the 'X'
-- component of our three-dimensional space. We must
-- convert the nautical mile difference to feet to have the
-- same units for later calculations.
--
begin
        temp_1 := range_xy *

Numerical_Algorithms.cos(Potential_Threat.get_relative_bearing(threat)) +
                (pt_velocity_xy *

Numerical_Algorithms.cos(Potential_Threat.get_ground_track(threat)) -
                ha_velocity_xy *
                  Numerical_Algorithms.cos(Host_Aircraft.get_ground_track)) *
                      (time / 3600.0);
exception
    when constraint_error =>
        text_io.put_line("sd - 25 CE ** " &
Potential_Threat.get_aircraft_id(threat));
    when numeric_error =>
        text_io.put_line("sd - 25 NE ** " &
Potential_Threat.get_aircraft_id(threat));
    when others =>
        text_io.put_line("sd - 25 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
        temp_1 := temp_1 * Physical_Quantities.nautical_mile_to_feet;
--
```

```
-- temp_2 holds the relative difference between the
-- host aircraft and potential threat at the elapsed time in the 'Y'
-- component of our three-dimensional space. We must
-- convert the nautical mile difference to feet to have the
-- same units for later calculations.
--
begin
      temp_2 := range_xy *

Numerical_Algorithms.sin(Potential_Threat.get_relative_bearing(threat)) +
                (pt_velocity_xy *

Numerical_Algorithms.sin(Potential_Threat.get_ground_track(threat)) -
                ha_velocity_xy *
                  Numerical_Algorithms.sin(Host_Aircraft.get_ground_track)) *
                    (time / 3600.0);
exception
   when constraint_error =>
      text_io.put_line("sd - 26 CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error =>
      text_io.put_line("sd - 26 NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others =>
      text_io.put_line("sd - 26 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
      temp_2 := temp_2 * Physical_Quantities.nautical_mile_to_feet;
--
-- temp_3 holds the relative altitude difference (in feet) between the host
aircraft
-- and potential threat at the elapsed time. In our three-dimensional space,
-- it is the 'Z' component.
--
begin
      temp_3 := (Potential_Threat.get_altitude(threat) -
Host_Aircraft.get_altitude) +
                (Potential_Threat.get_climb_rate(threat) -
                        Host_Aircraft.get_climb_rate) * (time / 60.0);
exception
   when constraint_error =>
      text_io.put_line("sd - 27 CE ** " &
Potential_Threat.get_aircraft_id(threat));
   when numeric_error =>
      text_io.put_line("sd - 27 NE ** " &
Potential_Threat.get_aircraft_id(threat));
   when others =>
      text_io.put_line("sd - 27 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
end ;
--
-- The distance is then computed using
```

```
--
--          range = (temp_1**2 + temp_2**2 + temp_3**2) ** 0.5
--
begin
       return Numerical_Algorithms.sqrt(temp_1 * temp_1 + temp_2 * temp_2 +
                                                              temp_3 *
temp_3);
exception
   when constraint_error =>
       text_io.put_line("sd - 28 CE ** " &
Potential_Threat.get_aircraft_id(threat));
       text_io.put("Alt: ");
          put(float(Potential_Threat.get_altitude(threat)), aft=>1, exp=>0);
          text_io.new_line;
       text_io.put("CR: ");
          put(float(Potential_Threat.get_climb_rate(threat)), aft=>1, exp=>0):
          text_io.new_line;
       text_io.put("TR: ");
          put(float(Potential_Threat.get_range(threat)), aft=>1, exp=>0):
          text_io.new_line;
       text_io.put("RB: ");
          put(float(Potential_Threat.get_relative_bearing(threat)), aft=>1,
exp=>0);
          text_io.new_line;
       text_io.put("GT: ");
          put(float(Potential_Threat.get_ground_track(threat)), aft=>1,
exp=>0);
          text_io.new_line;
       text_io.put("Vel: ");
          put(float(Potential_Threat.get_velocity(threat)), aft=>1, exp=>0);
          text_io.new_line;
       text_io.put("pt_velocity_xy: ");
          put(float(pt_velocity_xy), aft=>1, exp=>0);
          text_io.new_line;
       text_io.put("temp_1: ");
          put(temp_1, aft=>2, exp=>0);
          text_io.new_line;
       text_io.put("temp_2: ");
          put(temp_2, aft=>2, exp=>0);
          text_io.new_line;
       text_io.put("temp_3: ");
          put(temp_3, aft=>2, exp=>0);
          text_io.new_line;
       return 0.0;
   when numeric_error =>
       text_io.put_line("sd - 28 NE ** " &
Potential_Threat.get_aircraft_id(threat));
       return 0.0;
   when Numerical_Algorithms.root_negative =>
       text_io.put_line("sd - 28 sqrt error ** " &
Potential_Threat.get_aircraft_id(threat));
       return 0.0;
```

```
    when others =>
        text_io.put_line("sd - 28 Bozo error ** " &
Potential_Threat.get_aircraft_id(threat));
        return 0.0;
end ;

    end get_msd;

end Situation_Dynamics;
```

19. Temporary_Data_Buffers (TDB)

Spec

```
{
^module!external(tdb_spec)

^type(message_type, (module_name : target,
                        data_type : target))

^type(consumer_type, (consumer_name : target,
                        priority : target))

^program(tdb, (name : target.
                length : target,
                message : message_type,
                consumer : list of consumer_type
              ))

^module!internal(tdb_spec)

^prog_impl(tdb, spec, (
{
--
with {message.module_name};
package {name} is
}
^select(
    ^not(^member(consumer)) -> ( {
--
-- Add a message to the buffer in a first-in/first-out (FIFO) fashion. An
-- exception is raised if the message buffer would overflow resulting
-- in loss of data.
--
    procedure send(msg : in {message.module_name}.{message.data_type});
    {name}_Overflow : exception;


--
-- Remove the oldest message from the buffer (FIFO principle). The calling
-- program is suspended until a message is available.
--
    procedure receive(msg : out {message.module_name}.{message.data_type}); }
    )
    ^member(consumer) -> ( {
```

```
--
-- Message priority. These are listed in descending priority (high to low).
--
      type message_priority is ()
      ^forall(c, consumer, ( {
         {c.consumer_name}}
         ^select(
            ^not(^last(c)) -> ( {,}
            )
         )
      )){
      );


--
-- Add a message to the buffer (first-in/first-out principle) having
-- the specified priority. Different exceptions can be raised depending
-- on which buffer would overflow resulting in loss of data.
--
   procedure send(msg : in {message.module_name}.{message.data_type};
                    priority : in message_priority);}
   ^forall(c, consumer, ( {
   {name}_{c.consumer_name}_Overflow : exception;
}
))

      ^forall(c, consumer, ( {
--
-- Removes the oldest message from the buffer (FIFO principle) having
-- the priority {c.consumer_name}. The calling program is suspended until a
-- message is available. The request is processed only after
-- all higher priority requests have been processed first.
--
   procedure receive_{c.consumer_name}(msg : out
{message.module_name}.{message.data_type});}
      ))
   )
)
{

end {name};
}))
}
```

Body

```
{
^module!external(tdb_body)

^type(message_type, (module_name : target,
                     data_type : target))

^type(consumer_type, (consumer_name : target,
                      priority : target))
```

```
^program(restfoo, (length : target,
                   message : message_type,
                   pl : list of consumer_type))

^prog_impl(restfoo, v1, (
^select(
   ^member(pl) -> ( {
          or
            when}
            ^forall(x, pl, (
               ^select(
                  ^last(x) -> ( {
                   count_{x.consumer_name} > 0 =>
                 accept receive_{x.consumer_name}(msg : out
{message.module_name}.{message.data_type}) do
                   msg := data_buffer({x.consumer_name},
out_index_{x.consumer_name});
                 end;
                 out_index_{x.consumer_name} := out_index_{x.consumer_name} mod
{length} + 1;
                 count_{x.consumer_name} := count_{x.consumer_name} - 1;}
                  )
                  true -> ( {
                  count_{x.consumer_name} = 0 and then}
                  )
                )
            ))
            ^restfoo(length, message, ^filter(x, pl, ^not(^last(x))))
   )
)
))

^program(tdb, (name : target,
               length : target,
               message : message_type,
               consumer : list of consumer_type
               ))

^module!internal(tdb_body)

^prog_impl(tdb, body, (
{
with {message.module_name};
with System;
package body {name} is

--
-- Buffering task. Entries in the buffer are stored in a
-- first-in/first-out (FIFO) principle. The is a task entry
-- for sending a message to be buffered.}
^select(
   ^not(^member(consumer)) -> ( { There is also an
-- entry for receiving a message from the buffer.}
   )
```

```
    ^member(consumer) -> ( { There are also multiple
-- entries for receiving messages; one for each message
-- priority.}
    )
)
{
--
    task buffer is
}
^select(
    ^not(^member(consumer)) -> ( {
        entry send(msg : in {message.module_name}.{message.data_type});
        entry receive(msg : out {message.module_name}.{message.data_type});
        pragma Priority(12);}
    )
    ^member(consumer) -> ( {
        entry send(msg : in {message.module_name}.{message.data_type};
                    priority : in message_priority);}
        ^forall(c, consumer, ( {
            entry receive_{c.consumer_name}(msg : out
{message.module_name}.{message.data_type});}
        ))
{       pragma Priority(7);
}
    )
)
{

    end buffer;

}
^select(
    ^not(^member(consumer)) -> ( {
--
-- Add a message to the buffer in a first-in/first-out (FIFO) fashion.
--
        procedure send(msg : in {message.module_name}.{message.data_type})
        is
        begin
            buffer.send(msg);
        end send;}
    )
    ^member(consumer) -> ( {
--
-- Add a message to the buffer (first-in/first-out principle) having
-- the specified priority.
--
        procedure send(msg : in {message.module_name}.{message.data_type};
                    priority : in message_priority)
        is
        begin
            buffer.send(msg, priority);
        end send;}
```

```
      )
  )

  ^select(
      ^not(^member(consumer)) -> ( {
  --
  -- Remove the oldest message in the buffer. The calling program
  -- is suspended until a message is available.
  --
      procedure receive(msg : out {message.module_name}.{message.data_type})
      is
      begin
          buffer.receive(msg);
      end receive;}
      )
      ^member(consumer) -> (
          ^forall(c, consumer, ( {
  --
  -- Removes the oldest message from the buffer (FIFO principle) having
  -- the priority {c.consumer_name}. The calling program is suspended until a
  -- message is available. The request is processed only after
  -- all higher priority requests have been processed first.
  --
              procedure receive_{c.consumer_name}(msg : out
  {message.module_name}.{message.data_type})
              is
              begin
                  buffer.receive_{c.consumer_name}(msg);
              end receive_{c.consumer_name};}
          ))
      )
  )
  {

  --
  -- Task body.
  }
  ^select(
      ^not(^member(consumer)) -> ( {-- Messages are stored in a
  -- data_buffer having a fixed length.}
      )
      ^member(consumer) -> ( {
  -- Messages are stored in
  -- data_buffers; each buffer used for storing messages
  -- of a specified priority.}
      )
  )
  {
  --
      task body buffer is
  }
  ^select(
      ^not(^member(consumer)) -> ( {
```

```
        data_buffer : array(1..{length}) of
{message.module_name}.{message.data_type}:
        count : integer range 0 .. {length} := 0;
        in_index, out_index : integer range 1 .. {length} := 1;}
    )
    ^member(consumer) -> ( {
        data_buffer : array(message_priority, 1..{length}) of
{message.module_name}.{message.data_type};}
        ^forall(c, consumer, ( {
            count_{c.consumer_name} : integer range 0 .. {length} := 0;
            in_index_{c.consumer_name} : integer range 1 .. {length} := 1;
            out_index_{c.consumer_name} : integer range 1 .. {length} := 1;
}
        ))
    )
)
{
        temp_message : {message.module_name}.{message.data_type};
}
    ^select(
        ^member(consumer) -> ( {
        temp_priority : message_priority;}
        )
    )
{
    begin
        loop
            select
}
^select(
    ^not(^member(consumer)) -> ( {
            accept send(msg : in {message.module_name}.{message.data_type}) do
                temp_message := msg;
            end;
            if count < {length} then
                data_buffer(in_index) := temp_message;
                in_index := in_index mod {length} + 1;
                count := count + 1;
            else
                raise {name}_Overflow;
            end if;}

    )
    ^member(consumer) -> ( {
            accept send(msg : in {message.module_name}.{message.data_type};
                        priority : in message_priority) do
                temp_message := msg;
                temp_priority := priority;
            end;
            if }
        ^forall(c, consumer, ( {
                temp_priority = {c.consumer_name} then
                if count_{c.consumer_name} < {length} then
```

```
                    data_buffe..temp_priority, in_index_{c.consumer_name}) :=
temp_message;

                    in_index_{c.consumer_name} := in_index_{c.consumer_name} mod
{length} + 1;

                    count_{c.consumer_name} := count_{c.consumer_name} + 1;
                else
                    raise {name}_{c.consumer_name}_Overflow;
                end if;}
          ^select(
            ^not(^last(c)) -> ( {
           elsif }
            )
          )
       ))
{
            end if;
}
    )
)
^select(
    ^not(^memDer(consumer)) -> ( {
         or
            when count > 0 =>
                accept receive(msg : out
{message.module_name}.{message.data_type}) do
                    msg := data_buffer(out_index);
                end;
                out_index := out_index mod {length} + 1;
                count := count - 1;}
    )
    ^member(consumer) -> (
       ^restfoo(length, message, consumer)
    )
)
{
         end select;
       end loop;
    end buffer;

end {name};
}
))
}
```

ADAPTABLE DOCUMENTATION COMPONENTS

1. Software Requirements Specification (SRS)

The parameterized implementation of the Software Requirements Specification (SRS) for the ATD/CWM domain is presented on the following pages.

Air Traffic Control / Collision Warning Monitor Software Requirements Specification
ATD/CWM-SRS-1.0: Volume 1 of 1

<revision.indicator> : <revision.date>

SOFTWARE REQUIREMENTS SPECIFICATION

FOR THE

ATD/CWM COMPUTER SOFTWARE CONFIGURATION ITEM

OF

THE AIR TRAFFIC CONTROL / COLLISION WARNING MONITOR SYSTEM

CONTRACT NO. <contract.contract_number>

CDRL SEQUENCE NO. <contract.CDRL_number>

Prepared for:

<contract.agency>

Prepared by:

Software Productivity Consortium
SPC Building
2214 Rock Hill Rd.
Herndon, VA 22070

Authenticated by _____   Approved by _____
          (Contracting agency)   (Contractor)

Date _____   Date _____

1. SCOPE

This section identifies the computer software configuration item (CSCI). which briefly states the purpose of the system, describes the role of the CSCI within the system, and summarizes the purpose and content of this software requirements specification (SRS).

1.1. Identification

This SRS establishes the require nents for the CSCI identified as:

- System title: < system.name >

- System mnemonic: < system.mnemonic >

- System Identification number: < system.id >

- CSCI title: Air-Traffic-Display / Collision-Warning-Monitor

- CSCI mnemonic: ATD/CWM

- CSCI number: XXXX

1.2. CSCI Overview

The < system.mnemonic > system monitors air traffic to detect collision warning situations within a surrounding surveillance area. The ATD/CWM CSCI will provide the following capabilities:

- Potential_Threat monitoring. Monitors potential threat flight characteristics ground track. relative bearing. range altitude. airspeed. and climb rate within the surveillance area.

- Intersection monitoring. Moni.ors the probable intersection of all aircraft with the host aircraft.

- Collision warning situation detection. Detects collision warning situations with respect to each potential threat based upon its predicted flight path and the separation minima.

- Displays a corrective action advisory message on the host aircraft's display which describes what maneuvers the host aircraft should perform to avoid a collision.

< if alarm then >

- Sounds an audible alarm within the host aircraft's cockpit for a detected collision warning situation.

< endif >
< if inter_air_msg then >

- Transmits messages to the nearby potential threat for a detected collision warning situation.

< endif >

< if atc_msg then >

- Transmits a message to a nearby air traffic control center for a detected collision warning situation.

< endif >

1.3. Document Overview

This specification establishes engineering and qualifications requirements for the ATD/CWM CSCI and provides the software requirements allocated from the @@ for the CSCI. The engineering requirements include external interface and capability requirements, internal interface descriptions, and other CSCI requirements. The external interface requirements identify all interfaces between this CSCI and other CSCIs and between this CSCI and hardware configuration items (HWCI) or critical items. The capability requirements state inputs, processing, and outputs for each CSCI capability. The internal interface descriptions identify and briefly describe each of the interfaces between CSCI capabilities. The other requirements include requirements for data elements, adaptation, sizing and timing, safety, security, design constraints, software quality factors, and human performance/human engineering. Requirements traceability matrices map these requirements to corresponding requirements in the @@ and vice versa.

The qualification requirements describe the qualification methods to be performed to verify the CSCI special qualification requirements. The method will be used to verify each requirement is shown in a verification traceability matrix.

The notes section lists abbreviations and acronyms used in this specification.

2. APPLICABLE DOCUMENTS

This section states document precedence and lists all documents referenced in this specification.

2.1. Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

MIL-STD-1815A-1983   Reference Manual For the Ada Programming Language

Copies of specifications, standards, drawings, and publications required by suppliers in connection with specified procurement functions should be obtained from the contracting agency or as directed by the contracting officer.

2.2. Non-Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

3. ENGINEERING REQUIREMENTS

This section contains the external interface and capability requirements for the ATD/CWM CSCI, and it identifies internal CSCI interfaces. It also contains requirements for CSCI data elements,

adaptation, sizing and timing, safety, security, design constraints, software quality factors, and human performance/human engineering.

Each requirement is identified uniquely by an {#} symbol at the end of the requirements. The mappings of these requirements to corresponding requirements in the @/@ and vice versa are shown in paragraph 3.12.

3.1. CSCI External Interface Requirements

The ATD/CWM CSCI will input and output data to the following external components:

- Navigation (NAV)

- Radar (RADAR)

< if alarm then >

- Audible_Alarm (AA)

< endif >
< if atc_msg OR inter_air_msg then >

- Communication (COMM)

< endif >

- Air_Traffic_Display (ATD)

- Air_Traffic_Control (ATC)

Figure 7-1 shows the ATD/CWM external interfaces. Each external interface shown in the diagram is described in the subsequent subparagraphs.
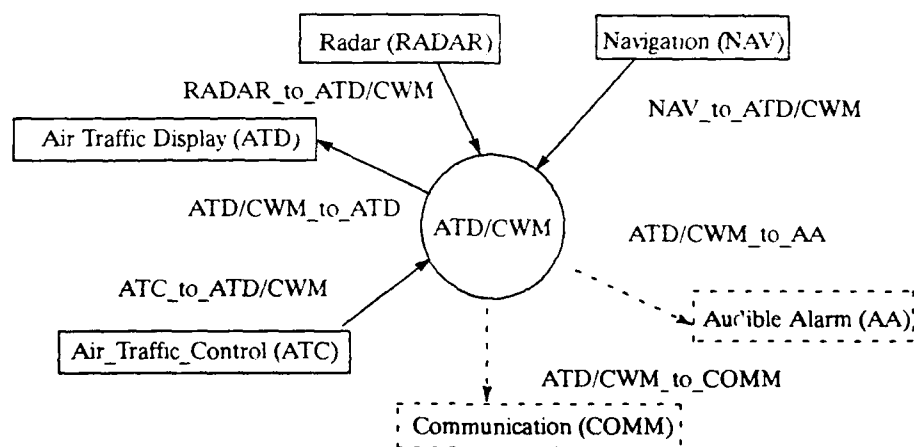
Note: Parameterization in this diagram is indicated by dashed lines (e.g., Audible Alarm).

Figure 7-1. Air Traffic Display/Collision Warning Monitor External Interface Diagram

3.1.1. RADAR_to_ATD/CWM Interface

The RADAR_to_ATD/CWM interface shall provide potential threat data from the RADAR to the ATD/CWM CSCI. This interface is specified in the Interface Requirements Specification {#}.

3.1.2. NAV_to_ATD/CWM Interface

The NAV_to_ATD/CWM interface shall provide host_aircraft data from the NAV to the ATD/CWM CSCI. This interface is specified in the Interface Requirements Specification {#}.

< if alarm then >

3.1.3. ATD/CWM_to_AA Interface

The ATD/CWM_to_AA interface shall provide a pitch and duration at which to ring an audible alarm. This interface is specified in the Interface Requirements Specification {#}.

< endif >

< if atc_msg OR inter_air_msg then >

3.1.4. ATD/CWM_to_COMM Interface

The ATD/CWM_to_COMM interface shall provide collision warning situation status from the ATD/CWM CSCI to the COMM. This interface is specified in the Interface Requirements Specification {#}.

< endif >

3.1.5. ATD/CWM_to_ATD Interface

The ATD/CWM_to_ATD interface shall provide collision warning situation status from the ATD/CWM CSCI to the ATD. This interface is specified in the Interface Requirements Specification {#}.

3.1.6. ATC_to_ATD/CWM Interface

The ATC_to_ATD/CWM interface shall provide potential_threat data from the ATC to the ATD/CWM CSCI. This interface is specified in the Interface Requirements Specification {#}.

3.2. CSCI Capability Requirements

The ATD/CWM CSCI will provide the following capabilities:

- Potential_Threat monitoring.

- Intersection monitoring.

- Collision warning situation detection.

3.3. CSCI Internal Interfaces

3.4. CSCI Data Element Requirements

3.5. Adaptation Requirements

This paragraph specifies the requirements for adapting the ATD/CWM CSCI to site-unique conditions and to changes in the system environment.

3.5.1. Installation-dependent Data

None.

3.5.2. Operational Parameters

3.6. Sizing and Timing Requirements

The ATD/CWM CSCI program storage shall not exceed 70 percent of the available memory {#}. The program storage capacity of the target computer is XXX.

3.7. Safety Requirements

3.8. Security Requirements

The ATD/CWM executable code shall be unclassified {#}.

3.9. Design Constraints

This paragraph specifies other requirements that constrain the CSCI design.

3.9.1. Programming Language.

The ATD/CWM CSCI shall be code in Ada and C. The Ada compiler is specified in MIL-STD-1815A-1983. The ATD/CWM CSCI Ada source code shall be compiled via the Ada compiler.

3.10. Software Quality Factors

This paragraph identifies the software quality factor requirements for the ATD/CWM CSCI.

3.11. Human Performance/Human Engineering Requirements

3.12. Requirements Traceability

4. QUALIFICATION REQUIREMENTS

This section specifies the qualification methods and any special qualification requirements necessary to establish that the ATD/CWM CSCI satisfies the requirements contained in Sections 3 and 5 of this specification.

4.1. Qualification Methods

To be determined.

4.2. Special Qualification Requirements

None

5. Preparation for Delivery

The source code shall be delivered on 8-track magnetic tape.

6. NOTES

This section contains information only and is not contractually binding.

6.1. Abbreviations and Acronyms

ATD   Air traffic display

CSCI   Computer software configuration item

CWM   Collision warning monitor

HWCI   Hardware configuration item

2. Interface_Requirements_Specification (IRS)

The parameterized implementation of the Interface Requirements Specification (IRS) for the ATD/CWM domain is presented on the following pages.

Air Traffic Control / Collision Warning Monitor Interface Requirements Specification
ATD/CWM-IRS-1.0: Volume 1 of 1

<revision.indicator> : <revision.date>

INTERFACE REQUIREMENTS SPECIFICATION

FOR THE

THE AIR TRAFFIC CONTROL / COLLISION WARNING MONITOR SYSTEM

CONTRACT NO. <contract.contract_number>

CDRL SEQUENCE NO. <contract.CDRL_number>

Prepared for:

<contract.agency>

Prepared by:

Software Productivity Consortium
SPC Building
2214 Rock Hill Rd.
Herndon, VA 22070

Authenticated by _____   Approved by _____
     (Contracting agency)   (Contractor)

Date _____   Date _____

1. SCOPE

This section identifies the interfaces, which briefly states the purpose of the system , describes the role of the interfaces within the system and summarizes the purpose and contents of this interface requirements specification (IRS).

1.1. Identification

This IRS establishes the requirements for the following interfaces:

- NAV_to_ATD/CWM

- RADAR_to_ATD/CWM

< if alarm then >

- ATD/CWM_to_AA

< endif >

< if atc_msg or inter_air_msg then >

- ATD/CWM_to_COMM

< endif >

- ATD/CWM_to_ATD

- ATC_to_ATD/CWM

These are the interfaces of < system.name > (< system.mnemonic >), < system.id > system. The subsequent subparagraphs list the computer software configuration items (CSCI) and hardware configuration items (HWCI) and critical items to which this IRS applies.

1.1.1. Applicable CSCIs

This IRS applies to the following CSCIs:

- ATD/CWM

1.1.2. Applicable HWCIs and Critical Items

This IRS applies to the interfaces between the CSCIs listed in the preceding subparagraph and the following HWCIs and critical items:

- Radar (RADAR)

- Navigation (NAV)

- Air_Traffic_Display (ATD)

- Air_Traffic_Control (ATC)

< if alarm then >

- Audible_Alarm (AA)

< endif >

< if atc_msg or inter_air_msg then >

- Communication (COMM)

< endif >

1.2. System Overview

The < system.mnemonic > system monitors air traffic within a surrounding surveillance area to detect collision warning situations. The following lists the role of each interface within the system:

- RADAR_to_ATD/CWM. Provides potential_threat data from the RADAR to the ATD/CWM CSCI.

- NAV_to_ATD/CWM. Provides host_aircraft data from the NAV to the ATD/CWM CSCI.

< if alarm then >

- ATD/CWM_to_AA. Provides a pitch and duration at which to ring an audible alarm from the ATD/CWM CSCI to the AA.

< endif >

< if atc_msg or inter_air_msg then >

- ∻ ATD/CWM_to_COMM. Provides collision warning situation status from the ATD/CWM CSCI to the COMM.

< endif >

- ATD/CWM_to_ATD. Provides collision warning situation status from the ATD/CWM CSCI to the ATD.

- ATC_to_ATD/CWM. Provides potential_threat data from the ATC to the ATD/CWM CSCI.

1.3. Document Overview

This specification establishes the detailed requirements for the interfaces between the applicable CSCIs and other configuration items. These detailed interface requirements have been allocated from the @@ and are referenced in software requirements specifications (SRS) of the applicable CSCIs.

The interface requirements describe interfaces between CSCIs and other CSCIs and between CSCIs and HWCIs and critical items. Requirements traceability matrices contained in the SRSs of the applicable CSCIs map these requirements to corresponding requirements in the @@.

The notes section lists abbreviations and acronyms used in this specification.

2. APPLICABLE DOCUMENTS

This section states document precedence and lists all documents referenced in this specification.

2.1. Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

MIL-STD-1815A-1983   Reference Manual For the Ada Programming Language

Copies of specifications, standards, drawings, and publications required by suppliers in connection with specified procurement functions should be obtained from the contracting agency or as directed by the contracting officer.

2.2. Non-Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

3. INTERFACE SPECIFICATION

This section specifies the interface requirements among CSCIs, HWCIs, and critical items to which this specification applies. The CSCI requirements to use the interfaces are specified in the SRS for each CSCI. The project-unique identifiers for the interfaces link the requirements in the SRSs to this IRS.

Table 7-1 shows the interfaces specified in this IRS by interfacing CSCI, HWCI, and critical item.

Table 7-1. Interface Relationships

CSCI, HWCI, or critical item	Interfaces with CSCI, HWCI, or critical item	Interface identifier

< if alarm then >

AA	ATD/CWM	ATD/CWM_to_AA

< endif >

ATC	ATD/CWM	ATC_to_ATD/CWM
ATD	ATD/CWM	ATD/CWM_to_ATD

< if alarm then >

ATD/CWM	AA	ATD/CWM_to_AA

< endif >

	ATC	ATC_to_ATD/CWM
	ATD	ATD/CWM_to_ATD

< if atc_msg or inter_air_msg then >

	COMM	ATD/CWM_to_COMM

< endif >

	NAV	NAV_to_ATD/CWM
	RADAR	RADAR_to_ATD/CWM

< if atc_msg or inter_air_msg then >

COMM	ATD/CWM	ATD/CWM_to_COMM

< endif >

NAV	ATD/CWM	NAV_to_ATD/CWM
RADAR	ATD/CWM	RADAR_to_ATD/CWM

3.1. Interface Diagrams

Figure 7-2 shows the interfaces for each applicable CSCI. Each interface specified in this IRS is described in the subsequent subparagraphs.
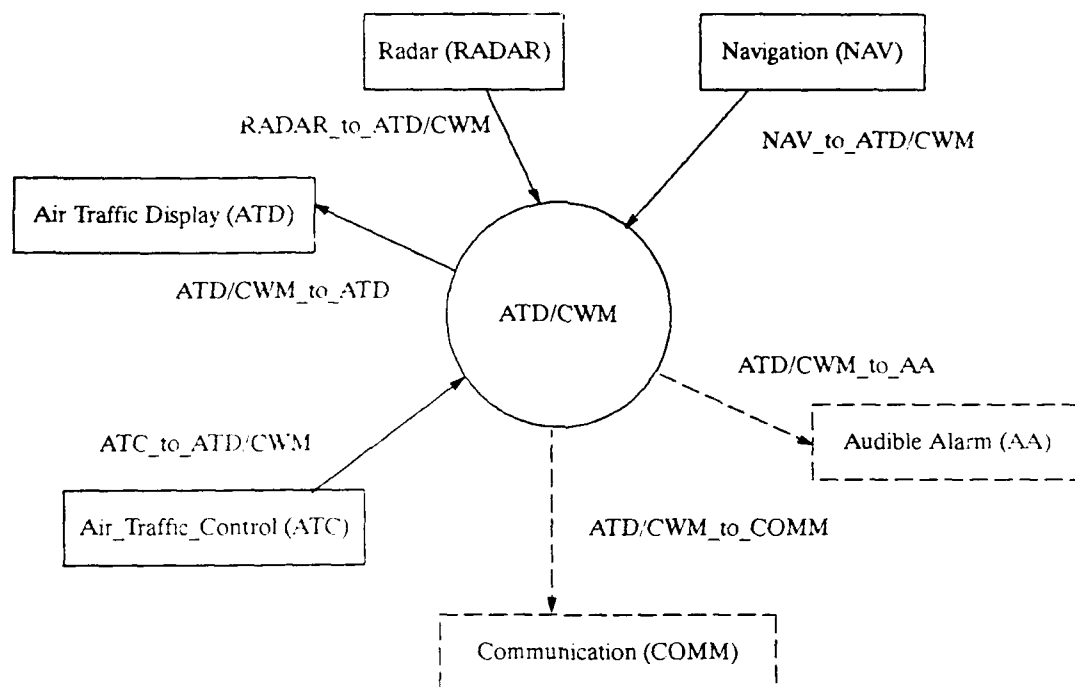
NOTE: Parameterization in this diagram is indicated by dashed lines (e.g.. Audible Alarm).

Figure 7-2. Air Traffic Display/Collision Warning Monitor External Interface Diagram

3.2. ATC_to_ATD/CWM Interface

The ATC_to_ATD/CWM interface provides potential_threat data from the ATC to the ATD/CWM CSCI.

3.2.1. Interface Requirements

The ATC_to_ATD/CWM data are transmitted via the serial data bus.

3.2.2. Data Requirements

Table 7-2 specifies an applicable information for the data elements transmitted across this interface. The source of these data elements is the ATC: their destination is the ATD/CWM CSCI.

Table 7-2. ATC_to_ATD/CWM Data Elements

Identifier	Description	Unit	Range	Accuracy	Precision
aircraft_id	Aircraft identification		N/A	N/A	N/A

Table 7-2, continued

Identifier	Description	Unit	Range	Accuracy	Precision
altitude	Vertical distance height of the potential_threat measured from mean sea level.	ft	0 to 60.000	1	1
velocity	Indicated airspeed of the host aircraft.	knots	0 to 700	1	1
relative bearing	Bearing of the potential threat relative to the host aircraft. Relative bearing is measured from the ground track of the host aircraft to the line from the host aircraft to the potential threat in the clockwise direction looking down.	degrees	0 to 360	0.1	0.1
range	Distance in nautical miles from this aircraft to the host_aircraft.	nm	0 to 300	0.1	0.1
ground_track	Ground_track is measured from the line of the aircraft to magnetic north to the horizontal component of the aircraft's actual flight path over the surface of the earth.	degrees	0 to 360	0.1	0.1
timestamp	Timestamp of when the data was valid. The timestamp is four digits representing the hours and minutes from the 24-hour clock.	HHMM	0000 to 2400	N A	N A

< if alarm then >

3.3. ATD/CWM_to_AA Interface

The ATD/CWM_to_AA interface provides pitch and duration from the ATD/CWM CSCI to the audible alarm.

3.3.1. Interface Requirements

The ATD/CWM_to_AA data are transmitted via the serial data bus.

3.3.2. Data Requirements

Table 7-3 specifies all applicable information for the data elements transmitted across this interface. The source of these data elements is the ATD/CWM CSCI; their destination is the AA.

Table 7-3. ATD/CWM_to_AA Data Elements

Identifier	Description	Unit	Range	Accuracy	Precision
pitch	Pitch of the audible_alarm in hertz.	Hz	1,000 to 10,000	1	1
duration	How long to ring the audible_alarm.	sec	0.01 to 10.0	0.01	0.01

< endif >

3.4. ATD/CWM_to_ATD Interface

The ATD/CWM_to_ATD interface provides collision warning situation status data from the ATD/CWM CSCI to the ATD.

3.4.1. Interface Requirements

The ATD/CWM_to_ATD data are transmitted via the serial data bus.

3.4.2. Data Requirements

Tables 7-4 and 7-5 specify all applicable information for the data elements transmitted across this interface. The source of these data elements is the ATD/CWM CSCI; their destination is the ATD.

Table 7-4. ATD/CWM_to_ATD Data Elements

Identifier	Description	Unit	Range	Accuracy	Precision
id	Handle for the displayed object.	N/A	0 to 32767	N/A	N/A
shape	Icon shape.	N/A	square: (1) circle: (2) triangle:(3)	N/A	N A
size	Size in pixels of the icon.	N/A	1 to 1000	N/A	N/A
fill	Color for the icon interior.	N/A	none: (1) yellow: (2) pink: (3) orange: (4) red: (5) green: (6) blue: (7) indigo: (8) purple: (9) violet: (10) black: (11) white: (12)	N/A	N/A

Table 7-4, continued

Identifier	Description	Unit	Range	Accuracy	Precision
color	Color for the icon.	N/A	none: (1) yellow: (2) pink: (3) orange: (4) red: (5) green: (6) blue: (7) indigo: (8) purple: (9) violet: (10) black: (11) white: (12)	N/A	N/A
fill_blink_rate	Blinking rate for the icon interior.	sec	0.0 to 10.0	0.1	0.1
obj_blink_rate	Blinking rate for the icon.	sec	0.0 to 10.0	0.1	0.1
x_location	Horizontal pixel location for icon center.	N/A	0 to 1100	1	1
y_location	Vertical pixel location for the icon center.	N/A	0 to 1100	1	1

Table 7-5. ATD/CWM_to_ATD Data Elements

Identifier	Description	Unit	Range	Accuracy	Precision
text	A variable length message describing what actions the pilot should perform to avoid a potential collision.	N/A	N/A	N/A	N/A
x_location	Horizontal pixel location for icon center.	N/A	0 to 1100	1	1
y_location	Vertical pixel location for the icon center.	N/A	0 to 1100	1	1

< if atc_msg or inter_air_msg then >

3.5. ATD/CWM_to_COMM Interface

The ATD/CWM_to_COMM interface provides ATC_Msg or Inter_Air_Msg messages from the ATD/CWM CSCI to the COMM.

3.5.1. Interface Requirements

The ATD/CWM_to_COMM data are transmitted via the serial data bus.

3.5.2. Data Requirements

Table 7-6 specifies all applicable information for the data elements transmitted across this interface. The source of these data elements is the ATD/CWM CSCI; their destination is the COMM.

Table 7-6. ATD/CWM_to_COMM Data Elements

< if atc_msg then >

Identifier	Description	Unit	Range	Accuracy	Precision
destination	Destination code.	N/A	1	N/A	N/A
code	The transponder code indicating the specific collision warning situation the host aircraft is in.	tc	0000 to 7777	N/A	N/A

< if mode = C then >

Identifier	Description	Unit	Range	Accuracy	Precision
altitude	Vertical distance height of the host aircraft measured from mean sea level.	ft	0 to 60,000	1	1

< endif >
< endif >

< if inter_air_msg then >

Identifier	Description	Unit	Range	Accuracy	Precision
destination	Destination code.	N/A	0	N/A	N/A
code	The transponder code indicating the specific collision warning situation the host aircraft is in.	tc	0000 to 7777	N/A	N/A
altitude	Vertical distance height of the host aircraft measured from mean sea level.	ft	0 t0 60,000	1	1
latitude	Latitude component of the location. Negative values represent latitude south of the equator.	degrees	-90 to 90	0.1	0.1
longitude	Longitude component of the location. A positive value signifies longitude west of the prime meridian at Greenwich, England. A negative value indicates longitude east of the prime meridian.	degrees	-360 to 360	0.1	0.1

< endif >
< endif >

3.6. NAV_to_ATD/CWM Interface

The NAV_to_ATD/CWM interface provides host aircraft status data from the NAV to the ATD/CWM CSCI.

3.6.1. Interface Requirements

The NAV_to_ATD/CWM data are transmitted via the serial data bus.

3.6.2. Data Requirements

Table 7-7 specifies all applicable information for the data elements transmitted across this interface. The source of these data elements is the NAV; their destination is the ATD/CWM CSCI.

Table 7-7. NAV_to_ATD/CWM Data Elements

Identifier	Description	Unit	Range	Accuracy	Precision
altitude	Vertical distance height of the host aircraft measured from mean sea level.	ft	0 to 60,000	1	1
velocity	Indicated velocity of the host aircraft.	knots	0 to 700	1	1
ground_track	Ground_track is measured from the line of the aircraft to magnetic north to the horizontal component of the aircraft's actual flight path over the surface of the earth.	degrees	0 to 360	0.1	0.1
latitude	Latitude component of the location. Negative values represent latitude south of the equator.	degrees	−90 to 90	0.1	0.1
longitude	Longitude component of the location. A positive value signifies longitude west of the prime meridian at Greenwich, England. A negative value indicates longitude east of the prime meridian.	degrees	−360 to 360	0.1	0.1

3.7. RADAR_to_ATD/CWM Interface

The RADAR_to_ATD/CWM interface provides potential_threat data from the RADAR to the ATD/CWM CSCI.

3.7.1. Interface Requirements

The RADAR_to_ATD/CWM data are transmitted via the serial data bus.

3.7.2. Data Requirements

Table 7-8 specifies all applicable information for the data elements transmitted across this interface. The source of these data elements is the RADAR; their destination is the ATD/CWM CSCI.

Table 7-8. RADAR_to_ATD/CWM Data Elements

Identifier	Description	Unit	Range	Accuracy	Precision
aircraft_id	Aircraft identification.	N/A	N/A	N/A	N/A
sweep	Radar sweep number module 32.	N/A	0 to 31 (modulo 32)	1	1
range	Distance in nautical miles from this aircraft to the host aircraft.	nm	0 to 300	0.1	0.00001525
relative bearing	Aircraft bearing relative to the host aircraft.	degrees	0 to 360	0.1	0.1

4. QUALITY ASSURANCE

None.

5. PREPARATION FOR DELIVERY

None.

6. NOTES

This section contains information only and is not contractually binding.

6.1. Abbreviations and Acronyms

< if alarm then > AA **< endif >**	Audible alarm
ATC	Air traffic control
ATD	Air traffic display
< if inter_air_msg OR atc_msg > COMM **< endif >**	Communication
CSCI	Computer software configuration item
CWM	Collision warning monitor
ft	Feet
HHMM	Four digit time (24–hour clock). Leftmost two digits (HH) are hours: the rightmost two digits (MM) are minutes.
HWCI	Hardware configuration item
Hz	Hertz
N/A	Not applicable
NAV	Navigation
nm	Nautical_miles
sec	Seconds
tc	Transponder code. Each of the four digits only having the range 0 .. 7.

This page intentionally left blank.

3. Software Design Document (SDD)

Note: The adaptable Software Design Document for the ATD/CWM domain has been purposely omitted to reduce the size of the ATD/CWM case study documentation. Refer to the adaptable SRS and adaptable IRS documents for other examples of adaptable documentation.

This page intentionally left blank.

ADAPTABLE VERIFICATION AND VALIDATION SUPPORT COMPONENTS

Adaptable CSU Tests

1. Audible_Alarm (AA)

1.1. Test Procedure and Results

The test suite for the unit test of Audible_Alarm consists of six modules of which four were written specifically for this unit test:

- aa_csu.trf – Abstract driver program

- aad_.a – Modified Ada package spec for the Audible_Alarm_Device module

- aad.a – Modified Ada package body for the Audible_Alarm_Device module

- pt_.trf – Abstract Potential_Threat package spec

The other two modules (aa.a and aa_.a) are the modules being tested. The test data for this unit test is comprised of the pitch and duration at which to ring the audible alarm for a specified collision warning situation as shown below.

Test Data

Collision Warning Situation	Frequency	Duration
^forall(c, ring. ({		
{r.cws_name}	{r.frequency}	{r.duration}}
))		

For each collision warning situation defined above, the test driver program writes the name of the collision warning situation to the screen. This program then gives the Audible_Alarm module a collision warning situation name. Audible_Alarm, in turn, gives the Audible_Alarm_Device (AAD) module the pitch and duration at which to ring the audible_alarm. Module AAD will write out to the screen these values. For each test case, the following outputs are expected.

```
^forall(c, ring. ( {
    {r.cws_name}            {r.frequency}      {r.duration}}
))
```

The test passes if the values produced by running the test exactly match these values for each collision warning situation.

1.2. Source Code

Source code written specifically to perform unit testing of module AA is shown below.

Audible_Alarm Driver

```
{
^module!external(aa_csu)

^type(ring_info, (cws_name : target,
                  frequency : target,
                  duration : target))

^program(aa_csu, (ring : list of ring_info))
}

{
^module!internal(aa_csu)

^prog_impl(aa_csu, v1, (
{
--
-- Driver for the Audible_Alarm module
--
with Potential_Threat;
with Text_IO; use Text_IO;
with Audible_Alarm;
procedure AA_CSU
is
begin
   put_line("CSU unit testing for module Audible_Alarm (AA)");
   new_line;
   put_line("This module will call module Audible_Alarm to see if");
   put_line("it correctly calls module Audible_Alarm_Device with the");
   put_line("proper frequency and duration for each corresponding");
   put_line("collision warning situation");
   new_line;
   put_line("The following frequency and duration are expected:");
   new_line;
   put_line("   Collision Warning Situation      Frequency      Duration");}
^forall(r, ring, ( {
   put_line("       {r.cws_name}                        {r.frequency}
{r.duration}");}
))
{
   new_line;

   put_line("These expected values must match those passed to
Audible_Alarm_Device");
   put_line("for each collision warning situation, respectively");
   new_line;
   put_line("Start of test");
   new_line;
   put_line("   Collision Warning Situation      Frequency      Duration");
```

```
}
^forall(r, ring, ( {
   put("       {r.cws_name}");
   Audible_Alarm.ring_alarm(Potential_Threat.{r.cws_name});}
))
{
   new_line(2);
   put_line("End of test");
end AA_CSU;
}
))
}
```

Audible_Alarm_Device (spec)

```
--
-- Audible_Alarm_Device (AAD) spec
--
-- This module is solely for the unit test of
-- module Audible_Alarm.
--
package Audible_Alarm_Device is

   type Duration is delta 0.01 range 0.01 .. 10.00:    -- seconds
   type Frequency is range 1000 .. 10_000;             -- hertz

   procedure ring_alarm(f : in Frequency;
                        d : in Duration);

end Audible_Alarm_Device;
```

Audible_Alarm_Device (body)

```
--
-- Audible_Alarm_Device (AAD) body
--
-- This module is used solely for the unit test
-- of the Audible_Alarm module.
--
with Text_IO;
package body Audible_Alarm_Device is

   package Duration_IO is new Text_IO.Fixed_IO(Duration); use Duration_IO;
   package Frequency_IO is new Text_IO.Integer_IO(Frequency); use
Frequency_IO;

   procedure ring_alarm(f : in Frequency;
                        d : in Duration)
   is
   begin
      text_io.put("                         ");
      put(f);
      text_io.put("        ");
      put(d);
```

```
        text_io.new_line;
    end ring_alarm;

end Audible_Alarm_Device;
```

Potential_Threat (TRF spec)

```
{
^module!external(pt_spec)

^type(cws_info, (cws_name : target))

^program(pt, (cws : list of cws_info))
}

{
^module!internal(pt_spec)

^prog_impl(pt, spec, (
{
--
-- Potential_Threat (PT)  package spec
--
-- This module is used solely for the unit test
-- of the Audible_Alarm module.
--
package Potential_Threat is

    type cws_id is (
}
^forall(c, cws, ( {
      {c.cws_name}, }
))
{
      normal
    );

end Potential_Threat;
}
))
}
```

2. Collision_Warning_Situation_Status (CWSS)

2.1. Test Procedure and Results

The test suite for the unit test of Collision_Warning_Situation_Status consists of nine modules of which four were written specifically for this unit test:

- cwss_csu.trf – Abstract driver program

- sd.a – Modified Ada package body for the Situation_Dynamics module

- pt_.trf – Abstract Potential_Threat package spec

- pt.a – Modified Ada package body for the Potential_Threat module

The following three modules needed for this test are used without any changes:

- sd_.a – Ada package spec for the Situation_Dynamics module

- pq_.a – Ada package spec for the Physical_Quantities module

- pq.a – Ada package body for the Physical_Quantities module

The remaining two modules (cwss_.a and cwss.a) are the modules being tested. The test data for this unit test is generated automatically by the test driver program (cwss_csu.trf) from the collision warning situation definitions provided as instantiation parameters to this module. The tests check the following scenarios:

- Boundary conditions in terms of time, range, or both (as appropriate) for each collision warning situation

- The correct collision warning situation detected as a function of the aircraft's partition

The following data is output for each test:

- Test number – Means for identifying the test

- Time – Predicted elapsed time before the host aircraft and this potential threat reach the predicted closest range

- Range – Distance the potential threat is from the host aircraft

- Partition – Potential threat aircraft partition

- Expected status – Predicted collision warning situation status for the potential threat based upon the time, range, and partition specified above

- Actual status – Collision warning situation status computed by module CWSS for the potential threat

If the "Expected status" and "Actual status" agree, then the following message is printed.

***** Correct status : Test <Test Number> passed *****

If they disagree, then the following message is printed.

***** Wrong status : Test <Test Number> failed *****

After all tests have been run, a test summary is printed describing the total number of tests that failed.

2.2. Source Code

Source code written specifically to perform unit testing of module CWSS is shown below.

Collision_Warning_Situation_Status Driver

```
{
^module!external(cwss_csu)

^type(time_type, (min : target,
                  max : target))

^type(range_type, (min : target,
                   max : target))

^type(t_and_r_type, (t_min : target,
                     t_max : target,
                     r_min : target,
                     r_max : target))

^type(cws_def, (time ?: time_type,
                range ?: range_type,
                t_and_r ?: t_and_r_type))

^type(cws_type, (cws_name : target,
                 severity : target,
                 predicate : cws_def,
                 partition : target))

^program(cwss_csu, (cws : list of cws_type,
                    area : target))
}

{
^module!internal(cwss_csu)

^program(array_length, (cws : list of cws_type))

^prog_impl(array_length, v1, (
    ^stream!int(s, (
        ^forall(x, ^filter(y, ^transpose((a:cws, b:s)), ^last(y)), (x.b
        ))
    ))
))
```

```ada
^prog_impl(cwss_csu, body, (
{
--
-- Test driver for CWSS unit testing.
--
with Potential_Threat; use Potential_Threat;
with Collision_Warning_Situation_Status;
with Text_IO; use Text_IO;
with Physical_Quantities; use Physical_Quantities;
procedure CWSS_CSU
is
    status : Potential_Threat.cws_id;
    errors : natural := 0;
    test_number : natural := 0;

    type local_partition is (L_ALL, L_UID, L_ID);

    type test_case;
    type test_case_ptr is access test_case;

    type test_case is
        record
            test_number : natural:
            expected_cws : Potential_Threat.cws_id;
            elapsed_time : Physical_Quantities.seconds;
            target_distance : Physical_Quantities.nautical_mile;
            partition : local_partition;
            next : test_case_ptr;
        end record;

    test_case_list : test_case_ptr;

    type cws_criteria is (time_only, range_only, time_and_range);

    type raw_data is
        record
            cws : Potential_Threat.cws_id;
            severity : float;
            partition : local_partition;
            predicate : cws_criteria;
            time_min : Physical_Quantities.seconds;
            time_max : Physical_Quantities.seconds;
            range_min : Physical_Quantities.nautical_mile;
            range_max : Physical_Quantities.nautical_mile;
        end record;

    pt : Potential_Threat.pt_handle;
    test : test_case_ptr;

    package seconds_io is new float_io(seconds); use seconds_io;
    package nautical_mile_io is new float_io(nautical_mile); use
nautical_mile_io;


--
-- Construct the test data based upon the collision warning
```

```
-- situation predicates and their respective partitions.
--
    procedure construct_data
    is
        maximum_time : Physical_Quantities.seconds;
        maximum_range : Physical_Quantities.nautical_mile;

        other_partition : local_partition;

        rd : array(1..{^array_length(cws)}) of raw_data := ()
        ^forall(c, cws, ( {
            ({c.cws_name}, {c.severity}, }
            ^select(
                ^equal(c.partition, {ALL}) -> ( {L_ALL, })
                ^equal(c.partition, {UID}) -> ( {L_UID, })
                ^equal(c.partition, {ID})  -> ( {L_ID, })
            )
            ^select(
                ^defined(c.predicate.time) -> ( {
                    time_only, {c.predicate.time.min}, {c.predicate.time.max}, 0.0,
0.0)}
                )
                ^defined(c.predicate.range) -> ( {
                    range_only, 0.0, 0.0, {c.predicate.range.min},
{c.predicate.range.max})}
                )
                true -> ( {
                    time_and_range, {c.predicate.t_and_r.t_min},
{c.predicate.t_and_r.t_max},
                    {c.predicate.t_and_r.r_min}, {c.predicate.t_and_r.r_max})}
                )
            )
            ^select(
                ^not(^last(c)) -> ( {,} )
                ^last(c) -> ( {);} )
            )
        ))
{

--
-- Make a test. If the test applies to all aircraft
-- partitions, then add duplicate copies of the test; one for
-- each partition.
--
        procedure make_test(cws : in Potential_Threat.cws_id;
                            time : in Physical_Quantities.seconds;
                            distance : in Physical_Quantities.nautical_mile;
                            partition : in local_partition)
        is
            p, q1, q2 : test_case_ptr;
        begin
            test_number := test_number + 1;
            p := new test_case'(test_number, cws, time, distance, partition,
```

```
null);
        if test_case_list = null then
            test_case_list := p;
        else
            q1 := null;
            q2 := test_case_list;
            while q2 /= null loop
                q1 := q2;
                q2 := q2.next;
            end loop;
            q1.next := p;
        end if;
    end make_test;
--
-- Make a test for both partitions using the supplied data
--
    procedure make_both_tests(cws : in Potential_Threat.cws_id;
                              time : in Physical_Quantities.seconds;
                              distance : in
Physical_Quantities.nautical_mile)
    is
    begin
        make_test(cws, time, distance, L_ID);
        make_test(cws, time, distance, L_UID);
    end make_both_tests;


--
-- Determine which collision warning situation will apply
-- to the partition given the specified time. If no other
-- time predicate applies, then generate tests for all
-- comparable range predicates.
--
    procedure scan_time_cws(time : in Physical_Quantities.seconds;
                            partition : in local_partition)
    is
        flag : boolean;
    begin
        flag := false;
        for x in rd'range loop
            if rd(x).predicate = time_only then
                if rd(x).partition = partition and then
                        rd(x).time_min <= time and then time < rd(x).time_max
then
                    flag := true;
                    make_test(rd(x).cws, time, 0.0, partition);
                    exit when flag = true;
                end if;
            end if;
        end loop;
        if flag = false then
            for x in rd'range loop
                if rd(x).predicate = range_only and then
                        (rd(x).partition = partition or else rd(x).partition =
```

```
L_ALL) then
                    make_test(rd(x).cws, time, rd(x).range_min, partition);
                    make_test(rd(x).cws, time, rd(x).range_max - 0.1,
partition);
                    make_test(rd(x).cws, time, (rd(x).range_max +
rd(x).range_min) / 2.0, partition);
                    flag := true;
                end if;
            end loop;
            if flag = false and then maximum_range < {area} then
                make_test(normal, time, maximum_range, partition);
                make_test(normal, time, {area} - 0.1, partition);
                make_test(normal, time, ({area} + maximum_range) / 2.0,
partition);
            end if;
        end if;
    end scan_time_cws;


--
-- Determine which collision warning situation will apply
-- to the partition given the specified range. If no other
-- range predicate applies, then generate tests for all
-- comparable range predicates.
--
    procedure scan_range_cws(distance : in
Physical_Quantities.nautical_mile;
                             partition : in local_partition)
    is
        flag : boolean;
    begin
        flag := false;
        for x in rd'range loop
            if rd(x).predicate = range_only then
                if rd(x).partition = partition and then
                        rd(x).range_min <= distance and then distance <
rd(x).range_max then
                    flag := true;
                    make_test(rd(x).cws, maximum_time, distance, partition);
                    exit when flag = true;
                end if;
            end if;
        end loop;
        if flag = false then
            for x in rd'range loop
                if rd(x).predicate = time_only and then
                        (rd(x).partition = partition or else rd(x).partition =
L_ALL) then
                    make_test(rd(x).cws, rd(x).time_min, distance, partition);
                    make_test(rd(x).cws, rd(x).time_max - 0.1, distance,
partition);
                    make_test(rd(x).cws, (rd(x).time_max + rd(x).time_min) /
2.0, distance, partition);
                    flag := true;
```

```
                    end if;
                end loop;
              end if;
          end scan_range_cws;

     begin
        test_case_list := null;
--
-- Find the largest range and largest time specifying
-- a collision warning situation.
--
        maximum_range := 0.0;
        maximum_time := 0.0;
        for x in rd'range loop
            if rd(x).predicate = range_only and then rd(x).range_max >
maximum_range then
                maximum_range := rd(x).range_max;
            elsif rd(x).predicate = time_only and then rd(x).time_max >
maximum_time then
                maximum_time := rd(x).time_max;
            end if;
        end loop;
--
-- Cycle through all partitions from highest severity to lowest severity
-- generating appropriate test data for each.
--
        for x in rd'range loop
            if rd(x).predicate = time_only then
                if rd(x).partition = L_ALL then
                    make_both_tests(rd(x).cws, rd(x).time_min, 0.0);
                    make_both_tests(rd(x).cws, rd(x).time_max - 0.1, 0.0);
                    make_both_tests(rd(x).cws, (rd(x).time_max + rd(x).time_min) /
2.0, 0.0);
                else
--
-- Collision warning situation does not apply to both partitions. Thus, we
need
-- to generate test data to ensure that this predicate does not apply to
-- the other partition.
--
                    make_test(rd(x).cws, rd(x).time_min, 0.0, rd(x).partition);
                    make_test(rd(x).cws, rd(x).time_max - 0.1, 0.0,
rd(x).partition);
                    make_test(rd(x).cws, (rd(x).time_max + rd(x).time_min) / 2.0,
0.0, rd(x).partition);
                    other_partition := L_ID;
                    if rd(x).partition = L_ID then
                        other_partition := L_UID;
                    end if;
--
-- Scan through all the predicates (from highest severity to lowest severity)
-- to see if any other applicable predicates are true. If so, then
-- generate a test for it.
```

```
--
                  scan_time_cws(rd(x).time_min, other_partition);
                  scan_time_cws(rd(x).time_max - 0.1, other_partition);
                  scan_time_cws((rd(x).time_max + rd(x).time_min) / 2.0,
other_partition);
               end if;
           elsif rd(x).predicate = range_only then
              if rd(x).partition = L_ALL then
                  make_both_tests(rd(x).cws, maximum_time, rd(x).range_min);
                  make_both_tests(rd(x).cws, maximum_time, rd(x).range_max -
0.1);
                  make_both_tests(rd(x).cws, maximum_time, (rd(x).range_max +
rd(x).range_min) / 2.0);
               else
--
-- Collision warning situation does not apply to both partitions. Thus, we
need
-- to generate test data to ensure that this predicate does not apply to
-- the other partition.
--
                  make_test(rd(x).cws, maximum_time, rd(x).range_min,
rd(x).partition);
                  make_test(rd(x).cws, maximum_time, rd(x).range_max - 0.1,
rd(x).partition);
                  make_test(rd(x).cws, maximum_time, (rd(x).range_max +
rd(x).range_min) / 2.0, rd(x).partition);
                  other_partition := L_ID;
                  if rd(x).partition = L_ID then
                     other_partition := L_UID;
                  end if;
                  scan_range_cws(rd(x).range_min, other_partition);
                  scan_range_cws(rd(x).range_max - 0.1, other_partition);
                  scan_range_cws((rd(x).range_max + rd(x).range_min) / 2.0,
other_partition);
               end if;
            end if;
         end loop;
      end construct_data;

begin
   put_line("CSU unit test for module Collision_Warning_Situation_Status
(CWSS)");
   new_line;
   put_line("Module CWSS is called repeatedly to determine the collision
warning");
   put_line("situation status of a potential threat. The value returned by
this");
   put_line("module is compared against the expected status value.");
   new_line(2);
   construct_data;
   put_line("Start of tests");
   new_line;
   test := test_case_list;
```

```
      while test /= null loop
         new_line;
         put_line("Test number " & natural'image(test.test_number));
         put("   Time: ");
         put(test.elapsed_time, exp => 0);
         put_line(" seconds");
         put("   Range: ");
         put(test.target_distance, exp => 0);
         put_line(" nautical_miles");
         pt.time := test.elapsed_time;
         pt.target_range := test.target_distance;
         if test.partition = L_UID then
            pt.par := UID;
            put_line("   Partition: UID");
         elsif test.partition = L_ID then
            pt.par := ID;
            put_line("   Partition: ID");
         else
            put_line("   Partition: ALL");
         end if;
         put_line("   Expected status: " &
   Potential_Threat.cws_id'image(test.expected_cws));
         status := Collision_Warning_Situation_Status.get_cws_status(pt);
         put_line("   Actual status: " & Potential_Threat.cws_id'image(status));
         if (status /= test.expected_cws) then
            put_line("      ***** Wrong status : Test " &
   natural'image(test.test_number) & " failed *****");
            errors := errors + 1;
         else
            put_line("      ***** Correct status : Test " &
   natural'image(test.test_number) & " passed *****");
         end if;
         test := test.next;
      end loop;
      new_line(2);
      if (errors /= 0) then
         put_line("Test Summary : " & integer'image(errors) & " case(s) failed");
      else
         put_line("Test Summary: All test cases passed");
      end if;
   end CWSS_CSU;
   }
   ))
   }
```

Potential_Threat (TRF spec)

```
{
^module!external(pt_spec)


^type(cws_info, (cws_name : target))
```

```
^program(pt, (cws : list of cws_info))
}


{
^module!internal(pt_spec)

^prog_impl(pt, spec, (
{
--
-- Potential_Threat (PT)  package spec
--
-- This module is used solely for the unit test
-- of the Collision_Warning_Situation_Status  module.
--
with Physical_Quantities;
package Potential_Threat is

    type partition is (ID, UID);

    type pt_handle is
       record
          time : Physical_Quantities.seconds;
          target_range : Physical_Quantities.nautical_mile;
          par : partition;
       end record;

    type cws_id is (
}
^forall(c, cws, ( {
      {c.cws_name}, }
))
{
      normal
    );

    function get_range(pt : in pt_handle) return
Physical_Quantities.nautical_mile;

    function get_partition(pt : in pt_handle) return partition;

end Potential_Threat;
}
))
}
```

Potential_Threat (body)

```
with Physical_Quantities;
package body Potential_Threat is

    function get_range(pt : in pt_handle) return
Physical_Quantities.nautical_mile
    is
    begin
```

```
         return pt.target_range;
      end get_range;

      function get_partition(pt : in pt_handle) return partition
      is
      begin
         return pt.par;
      end get_partition;

   end Potential_Threat;
```

Situation_Dynamics (body)

```
   --
   -- Situation_Dynamics (SD) package body
   --
   with Physical_Quantities;
   with Potential_Threat;
   package body Situation_Dynamics is

      function get_elapsed_time(threat : in Potential_Threat.pt_handle)
                                          return Physical_Quantities.seconds
      is
      begin
         return threat.time;
      end get_elapsed_time;

      function get_msd(threat : in Potential_Threat.pt_handle)
                                          return Physical_Quantities.feet
      is
      begin
         return 0.0;
      end get_msd;

   end Situation_Dynamics;
```

This page intentionally left blank.

2. GENERATION PROCEDURE

Note: This Generation Procedure was written with the Consortium's computer environment in mind. There is a network consisting of Apollo and VAX/VMS computers. The adaptable code components used in this procedure reside on an Apollo; adapting these components also takes place on an Apollo. After the components have been adapted, they are transferred to an VAX running VMS. One mechanism for transferring the components is by using a Consortium-modified UNIX transfer program called **rcp.** Compiling, linking, and execution subsequently takes place on the VAX/VMS. An X-terminal client uses an Apollo node to simulate the ATD display for the ATD/CWM system.

The Generation Procedure describes how to generate a working system from the Product Implementation using the decision resolutions of the Application Model and the Decision Model Extensions. This Generation Procedure consists of four major steps:

1. Transforming the ATD/CWM Application Model into the canonic decision model form for ATD/CWM

2. Selecting Adaptable Components

3. Adapting the components

4. Composing a system from the Adapted Components

Each of these steps will be described in greater detail in the following sections.

Step 1. Application Model Transformation

You must first transform your validated ATD/CWM Application Model (from its external form) into an equivalent internal form expressed in terms of the ATD/CWM decision model before you proceed with the remaining activities of the Generation Procedure. The ATD/CWM decision model consists of the following decision classes:

- Aircraft_Status_Display

- Host_Aircraft_Status_Display

- Aircraft_Display_Symbol

- Collision_Warning_Situation_Response

- ATC_Message

- Collision_Warning_Situation

- Surveillance_Area

To do this transformation, you will need to fill in forms that are provided with each step. Each form has the following organization.

Form Name	Value
Decision Name	

The first column identifies the name of the form (boldface) and its related decisions. You use the second column to record values for the decisions. You derive these values from your external form of the ATD/CWM Application Model.

The transformation steps you must follow are listed below. You can perform these steps in any order. However, you must perform all of them before you have completed the internal form of the Application Model.

1. Fill in a Surveillance_Area form (Table 7-9). This form appears exactly once. Get the value for range from the Host_Aircraft Characteristics Surveillance_Area.

Table 7-9. Surveillance_Area

Surveillance_Area	Value
Range	

2. Fill in the Collision_Warning_Situation form (Table 7-10). This form is repeated once for every collision warning situation defined in your ATD/CWM Application Model. For example, if you have three collision warning situations, there will be three instances of this decision class. The values for each instance of this decision class are obtained using the following steps.

Table 7-10. Collision_Warning_Situation

Collision_Warning_Situation	Value
CWS_Name	
CWS_Def.Time.Min	
CWS_Def.Time.Max	
CWS_Def.Range.Min	
CWS_Def.Range.Max	
CWS.Partition	
Severity	
Response	

2.1. Get the value for CWS_Name from Collision Warning Situation CWS_Name.

2.2. Get the value for CWS_Def.Time.Min from Collision Warning Situation Time_Min. If no value for Time_Min is specified, leave CWS_Def.Time.Min blank.

2.3. Get the value for CWS_Def.Time.Max from Collision Warning Situation Time_Max. If no value for Time_Max is specified, leave CWS_Def.Time.Max blank.

2.4. ⋯ the value for CWS_Def.Range.Min from Collision Warning Situation Range_Min. If no value for Range_Min is specified, leave CWS_Def.Range.Min blank.

2.5. Get the value for CWS_Def.Range.Max from Collision Warning Situation Range_Max. If no value for Range_Max is specified, leave CWS_Def.Range.Max blank.

2.6. Get the value for CWS.Partition from Collision Warning Situation Partition.

2.7. Get the value for Severity from Collision Warning Situation Severity.

2.8. Get the value for Response by concatenating Collision Warning Situation CWS_Name with the text "_Response." For example, if the value for CWS_Name is Possible, the value for Response would be Possible_Response.

3. Fill in the Collision_Warning_Situation_Response form (Table 7-11). Repeat this form once for every collision warning situation defined in your ATD/CWM Application Model. For example, if you have three collision warning situations, there will be three instances of this decision class. Use the following steps to obtain the values for each instance of this decision class.

Table 7-11. Collision_Warning_Situation_Response

Collision_Warning_Situation_Response	Value
CWSR_Name	
ATC_Msg	
Inter_Air_Msg	
Corrective_Msg	
Alarm	
Alarm.Pitch	
Alarm.Duration	
Code	

3.1. Get the value for CWSR_Name by concatenating the value for Collision Warning Situation CWS_Name with the text "_Response". For example, if the value for CWS_Name is Possible, the value for Response would be Possible_Response.

3.2. Get the value for ATC_Msg from Collision Warning Situation ATC_Msg.

3.3. Get the value for Inter_Air_Msg from Collision Warning Situation Inter_Air_Msg.

3.4. Get the value for Corrective_Msg from Collision Warning Situation Corrective_Msg.

3.5. Get the value for Alarm from Collision Warning Situation Alarm.

3.6. Get the value for Alarm.Pitch from Collision Warning Situation Alarm_Pitch.

3.7. Get the value for Alarm.Duration from Collision Warning Situation Alarm_Duration.

3.8. Get the value for Code from Code.

4. Fill in the ATC_Message form (Table 7-12). This form appears exactly once. Get the value for Mode from Collision Warning Situation Message_Mode.

Table 7-12. ATC_Message

ATC_Message	Value
Mode	

5. Fill in the Aircraft_Status_Display form (Table 7-13). Repeat this form once for every collision warning situation that applies to a specific aircraft partition. For example, assume that your Application Model defines collision warning situations S1, S2, and S3. Furthermore, assume that S1 applies to all aircraft, S2 only applies to identified aircraft, and S3 only applies to unidentified aircraft. This would result in four instances of this decision class: two for S1, one for S2, and one for S3. Use the following steps to obtain the values of each instance of this decision class.

Table 7-13. Aircraft_Status_Display

Aircraft_Status_Display	Value
Situation	
Partition	
PT_Color	
PT_Blink	
PT_Fill	

5.1. Examine the value for Collision Warning Situation Partition for the current collision warning situation. If the value for mnemonic Partition is **ID** or **ALL**, you provide an additional Aircraft_Status_Display form to fill in. Then you must perform the following steps to obtain values for its related decisions.

5.1.1. Get the value for Situation from Collision Warning Situation CWS_Name.

5.1.2. The value for Partition is **ID**.

5.1.3. Get the value for PT_Color from Collision Warning Situation ID_Color.

5.1.4. Get the value for PT_Blink from Collision Warning Situation ID_Blink.

5.1.5. Get the value for PT_Fill from Collision Warning Situation ID_Fill.

5.2. If the value for Collision Warning Situation Partition is **UID** or **ALL**, you provide an additional Aircraft_Status_Display form to fill in. Then you must perform the following steps to obtain values for its related decisions.

5.2.1. Get the value for Situation from Collision Warning Situation CWS_Name.

5.2.2. The value for Partition is **UID**.

5.2.3. Get the value for PT_Color from Collision Warning Situation UID_Color.

5.2.4. Get the value for PT_Blink from Collision Warning Situation UID_Blink.

5.2.5. Get the value for PT_Fill from Collision Warning Situation UID_Fill.

6. Fill in the Host_Aircraft_Status_Display form (Table 7-14). Repeat this form once for every collision warning situation defined in your ATD/CWM Application Model. For example, if you have three collision warning situations, there will be three instances of this decision class. Use the following steps to obtain the values for each instance of this decision class.

Table 7-14. Host_Aircraft_Status_Display

Host_Aircraft_Status_Display	Value
Situation	
Color	

6.1. Get the value for Situation from Collision Warning Situation CWS_Name.

6.2. Get the value for Color from Collision Warning Situation Host_Color. The Host_Color must correspond to the named collision warning situation.

7. Fill in the Aircraft_Display_Symbol form (Table 7-15). This form appears exactly once.

Table 7-15. Aircraft_Display_Symbol

Aircraft_Display_Symbol	Value
Host_Shape	
ID_Shape.Shape	
ID_Shape.Partition	
UID_Shape	

7.1. Get the value for Host_Shape from Host_Aircraft_Characteristics Host_Shape.

7.2. Get the value for ID_Shape.Shape from Potential_Threat Characteristics ID_Shape.

7.3. Get the value for ID_Shape.Partition from Potential_Threat Characteristics ID_Req.

7.4. Get the value for UID_Shape from Potential_Threat Characteristics UID_Shape.

Step 2. Select the Adaptable Components

You will use the information you captured in Step 1 to select adaptable components.

Table 7-16 describes selection criteria for each adaptable component. The first column of the table names the concrete components that can potentially be included in a generated system. The second column describes the selection criteria for each component. You select the concrete component only when the criteria is True. An **Always** condition means that the component is always selected. References in the criteria correspond to decisions captured in Step 1. You select the concrete component only when the conditions are True. A third column has been added so that you can use Table 7-16 as a worksheet to indicate which components you have selected.

Below the selection criteria for each concrete component is a description of how the component is implemented. The first item is the names (can be more than one) of the text files that implement the concrete component followed by the implementation language in parentheses. Ada designates an Ada language component (these require no adaptation); Ada_Generic designates an Ada generic; TRF2 designates that the text file is written in a combination of TRF2 and Ada; C designates a C language component; Interleaf designates that the text file is written using Interleaf (i.e., a text processing tool). The text files (except those implemented in Interleaf) listed in the "Implemented By" column are located in the following Apollo directory.

//venus/local/public/atd_cwm_adaptable_components/code_components

The Interleaf documents are located in following directory.

//venus/local/public/atd_cwm_adaptable_components/doc_components

To select the adaptable component, you must evaluate the selection criteria for each concrete component. Record the names of the concrete components you selected so that you can adapt them where necessary in the next step.

As an example, you would select the concrete component named Audible_Alarm_Buffer only if at least one of the Collision Warning Situations from the Application Model internal form had a C.Response.Alarm value of True. On the other hand, the Concrete Component named Host_Aircraft would always be included in a generated system.

Table 7-16. Component Selection Criteria

Concrete Component Name	Include this Concrete Component...	✔
Audible_Alarm_Device	If there is a Collision Warning Situation, C, such that C.Response.Alarm is True. Implemented By: aad_.trf (TRF2)	
Audible_Alarm_Buffer	If there is (1) a Collision Warning Situation, C, such that C.Response.Alarm is True. Implemented By: tdb_.trf (TRF2) tdb.trf (TRF2)	
Communication_Device	If there is a Collision Warning Situation, C, such that either C.Response.ATC_Msg OR C.Response.Inter_Air_Msg is True. Implemented By: cd_.trf (TRF2) cd.trf (TRF2)	
Communication_Buffer	If there is (1) a Collision Warning Situation, C, such that either C.Response.ATC_Msg OR C.Response.Inter_Air_Msg is True. Implemented By: tdb_.trf (TRF2) tdb.trf (TRF2)	
Audible_Alarm	If there is a Collision Warning Situation, C, such that C.Response.Alarm is True. Implemented By: aa_.a (Ada) aa.trf (TRF2)	
Communication	If there is a Collision Warning Situation, C, such that either C.Response.ATC_Msg OR C.Response.Inter_Air_Msg is True. Implemented By: comm_.trf (TRF2) comm.trf (TRF2)	

Table 7-16, continued

Concrete Component Name	Include this Concrete Component...		✔
Radar_Target_Priority_Buffer	Always		✔
	Implemented By:	tdb_.trf (TRF2) tdb.trf (TRF2)	
Potential_Threat	Always		✔
	Implemented By:	pt_.trf (TRF2) pt.trf (TRF2)	
Target_Buffer	Always		✔
	Implemented By:	tdb_.trf (TRF2) tdb.trf (TRF2)	
Host_Aircraft	Always		✔
	Implemented By:	ha_.a (Ada) ha.a (Ada)	
Initialization_and_Termination	Always		✔
	Implemented By:	it.a (Ada)	
Navigation	Always		✔
	Implemented By:	nav_.a (Ada) nav.a (Ada)	
Radar	Always		✔
	Implemented By:	radar_.a (Ada) radar.a (Ada)	
Air_Traffic_Control	Always		✔
	Implemented By:	atc_.a (Ada) atc.a (Ada)	
Air_Traffic_Display_Device	Always		✔
	Implemented By:	atdd_.a (Ada) atdd.a (Ada) xlibrary.c (C)	
Collision_Warning_Situation_Status	Always		✔
	Implemented By:	cwss_.a (Ada) cwss.trf (TRF2)	

Table 7-16, continued

Concrete Component Name	Include this Concrete Component...	✔
Physical_Quantities	Always Implemented By: pq_.a (Ada) pq.a (Ada)	✔
Numerical_Algorithms	Always Implemented By: na_.a (Ada) na.a (Ada)	✔
Air_Traffic_Display	Always Implemented By: atd_.a (Ada) atd.a (Ada)	✔
Potential_Threat_Partition	If there is a Collision Warning Situation such that CWS.Partition is not ALL. Implemented By: ptp_.a (Ada_Generic) ptp.a (Ada_Generic)	
Situation_Dynamics	Always Implemented By: sd_.a (Ada) sd.a (Ada)	✔
Aircraft_Motion	Always Implemented By: am_.a (Ada_Generic) am.a (Ada_Generic)	✔
SRS	Always Implemented By: SRS.doc (Interleaf)	✔
IRS	Always Implemented By: IRS.doc (Interleaf)	✔

Step 3. Adapt the Components

Each of the adaptable code components is normally implemented by two parts: a specification and a body. You adapt either the specification, body, or both for a given adaptable code component as described in Table 7-16.

The form of the values for the adaptation parameters is subject to constraints imposed by the component implementation language. Examples of constraints include numeric precision and grammatical rules (e.g., capitalization). You must ensure that the constraints for the adaptable components are met; otherwise, you will not be able to produce the desired ATD/CWM system. You can assume

that the form of a value for a particular parameter has no constraints unless otherwise specifically expressed.

You adapt only those adaptable components you selected in Step 2. Thus, you will not necessarily follow all of the steps listed below.

Adapt Ada Generics

- To adapt Aircraft_Motion, you create a text file named am_gen.a which contains the following text verbatim.

  with Aircraft_Motion;
  package Air_Craft_Motion is new Aircraft_Motion(msd = > 500.0);

- To adapt Potential_Threat_Partition, you create a text file named pt_partition.a which contains the following text verbatim.

  with Potential_Threat;
  package PT_Partition is new Potential_Threat_Partition(altitude = > *Altitude*,
  airspeed = > *Airspeed*);

  You determine the values for *Altitude* and *Airspeed* by examining your Application Model internal form. *Altitude* has the value **true** if, and only if, the value for Aircraft_Display_Symbol ID_Shape.Partition contains the value of **altitude**. Otherwise, the value for *Altitude* is **false**.

  Similarly, *Airspeed* has the value **true** if, and only if, the value for Aircraft_Display_Symbol ID_Shape.Partition contains the value of **airspeed**. Otherwise, the value for *Airspeed* is **false**.

Adapt TRF2 Components

You mechanically adapt components written in TRF2 by using the TRF2 translator. The exact form and use of the TRF2 metaprogramming notation is described in the *TRF2 Metaprogramming Tool User Guide* (Software Productivity Consortium 1991b). For convenience, a **csh** script (named **adapt.csh**) has been provided which contains all of the translations possible in constructing a system. This script file is located in the following Apollo directory:

//venus/local/public/atd_cwm_adaptable_components/code_components

You must remove from this file all lines referencing components that are not to be included in your ATD/CWM system. The **csh** script file contains hardcoded file names that will be used in naming the resulting concrete components. These hardcoded names will be used in later scripts for controlling compilation. However, if so desired, you can rename the files contained in these scripts to any name as long as the names do not conflict with any implementation names listed in Table 7-16 nor conflict with any names chosen for the text files describing the adaptations of Ada generics. The TRF2 translator is located in the following Apollo file:

~spectrum/TRF/apollo.trft

You must create a single text file, called a metafile, for each part of an adaptable component to be adapted using TRF2 (e.g., Communication_Device requires two text files; Audible_Alarm only

requires one). This metafile contains the adaptation parameter values used by TRF2 in modifying the adaptable component. Templates for these metafiles (denoted by the suffix ".meta") are found in the following Apollo directory:

//venus/local/public/atd_cwm_adaptable_components/code_components

You should modify these templates according to the directions provided below. The names of the metafiles are hardcoded into the **adapt.csh** script. However, if so desired, you can change the names of these metafiles, and likewise the names in the script, to any name as long as the names do not conflict with any implementation names listed in Table 7-16 nor conflict with any names chosen for the text files describing the adaptations of Ada generics.

The contents of each TRF2 metafile is described below. You must modify each metafile according to the directions given to guarantee a valid adaptation of the component. You must replace the *boldface italicized* identifiers with the corresponding values from the internal form of the Application Model you developed in Step 1.

- **AA.TRF**—To adapt aa.trf for Audible_Alarm, you create a metafile called aa.meta which contains the following statements:

```
{ ^ module!include(aa_spec, "aa.trf")}
{ ^ aa_spec.aa!spec(
            ring : ( (    cws_name : {CWS_Name},
                          frequency : {Alarm.Pitch [see note 1]},
                          duration : {Alarm.Duration [see note 2]} ),
                    . . . see note 3
                  )
          )
}
```

Note 1: The value for *Alarm.Pitch* must be an integer (i.e., have no decimal point).

Note 2: The value for *Alarm.Duration* must have exactly two digits after the decimal point.

Note 3: The triple (cws_name, frequency, duration) is repeated once for every collision warning situation that has a "true" value for its Alarm mnemonic. Each triple is enclosed in parentheses and has a trailing comma except for the last triple in the list.

- **AAB_.TRF and AAB.TRF**—To adapt tdb_.trf for Audible_Alarm_Buffer, you create a metafile called aab_.meta which contains the following statements:

```
{ ^ module!include(tdb_spec, "tdb_.trf")}
{ ^ tdb_spec.tdb!spec(
            name : {Audible_Alarm_Buffer},
            length : {10},
            message : ( module_name : {Audible_Alarm_Device},
                        data_type : {Alarm_Message_Type}),
            consumer : ()
          )
}
```

To adapt tdb.trf for Audible_Alarm_Buffer, you create a metafile called aab.meta which contains the following statements:

```
{ ^ module!include(tdb_body, "tdb.trf")}
{ ^ tdb_body.tdb!body(
            name : {Audible_Alarm_Buffer},
            length : {10},
            message : ( module_name : {Audible_Alarm_Device},
                        data_type : {Alarm_Message_Type}),
            consumer : ()
    )
}
```

- **AAD_.TRF**—To adapt aad_.trf for Audible_Alarm_Device, you create a metafile called aad_.meta which contains the following statements:

```
{ ^ module!include(aad_spec, "aad_.trf")}
{ ^ aad_spec.aad!spec( {True})
}
```

- **CB_.TRF and CB.TRF**—To adapt tdb_.trf for Communication_Buffer, you create a metafile called cb_.meta which contains the following statements:

```
{ ^ module!include(tdb_spec, "tdb_.trf")}
{ ^ tdb_spec.tdb!spec(
            name : {Communication_Buffer},
            length : {10},
            message : ( module_name : {Communication_Device},
                        data_type : {Communication_Msg_Type}),
            consumer : ()
    )
}
```

To adapt tdb.trf for Communication_Buffer, you create a metafile called cb.meta which contains the following statements:

```
{ ^ module!include(tdb_body, "tdb.trf")}
{ ^ tdb_body.tdb!body(
            name : {Communication_Buffer},
            length : {10},
            message : ( module_name : {Communication_Device},
                        data_type : {Communication_Msg_Type}),
            consumer : ()
    )
}
```

- **CD_.TRF and CD.TRF**—To adapt cd.trf for Communication_Device, you create a metafile called cd.meta which contains the following statements:

```
{ ^ module!include(cd_body, "cd.trf")}
{ ^ cd_body.cd!body(   {see note 4}.
                       {see note 5},
                       {Mode [see note 6]},
                       {True})
}
```

To adapt cd_.trf for Communication_Device, you create a metafile called cd_.meta which contains the following statements:

```
{ ^ module!include(cd_spec, "cd_.trf")}
{ ^ cd_spec.cd!spec(   {see note 4},
                       {see note 5},
                       {Mode [see note 6]},
                       {True})
}
```

Note 4: If you have at least one collision warning situation which contains a "true" value for the ATC_Msg mnemonic, then this parameter's value is True; otherwise it is False. The capitalization, as indicated, must be used.

Note 5: If you have at least one collision warning situation which contains a "true" value for the Inter_Air_Msg mnemonic, then this parameter's value is True; otherwise it is False. The capitalization, as indicated, must be used.

Note 6: The value for *Mode* must be upper-case.

- **COMM_.TRF and COMM.TRF**—To adapt comm_.trf for Communication, you create a metafile called comm_.meta which contains the following statements:

```
{ ^ module!include(comm_spec, "comm_.trf")}
{ ^ comm_spec.comm!spec(
        (   (cws_name : {CWS_Name}, code : {Response.Code [see note 7]}),
            . . . see note 8
        ),
        (),
        {Mode [see note 9]})
}
```

To adapt comm.trf for Communication, you create a metafile called comm.meta which contains the following statements:

```
{ ^ module!include(comm_body, "comm.trf")}
{ ^ comm_body.comm!body(
        (   (cws_name : {CWS_Name}, code : {Response.Code [see note 7]}),
            . . . see note 8
        ),
        (),
        {Mode [see note 9]})
}
```

Note 7: The value for *Response.Code* must be an integer (i.e., have no decimal point).

Note 8: Repeat the pair (cws_name, code) once for every collision warning situation you have defined in Step 1 that also has a **true** value for either ATC_Msg or Inter_Air_Msg. Each ordered pair is enclosed in parentheses and has a trailing comma except for the last ordered pair in the list.

Note 9: The value for *Mode* must be all upper-case.

- **CWSS.TRF**—To adapt cwss.trf for Collision_Warning_Situation_Status, you create a metafile called cwss.meta which contains the following statements:

```
{ ^ module!include(cwss_body, "cwss.trf")}
{ ^ cwss_body.cwss!body(
        (  (  cws_name : {CWS_Name},
              severity : {Severity},
              predicate : see note 10,
              partition : {CWS.Partition}),
        . . . see note 11
        )
    )
}
```

Note 10: The predicate component of the ordered quadruple (cws_name, severity, predicate, partition) has one of the following forms depending on the situation flight characteristics for the named collision warning situation.

(time : (min : {*CWS_Def.Time.Min*}, max : {*CWS_Def.Time.Max*}))

(range : (min : {*CWS_Def.Range.Min*}, max: {*CWS_Def.Range.Max*}))

(t_and_r : (t_min : {*CWS_Def.Time.Min*},
            t_max: {*CWS_Def.Time.Max*},
            r_min: {*CWS_Def.Range.Min*},
            r_max: {*CWS_Def.Range.Max*}))

Use the first form when the situation flight characteristics for the named collision warning situation are specified by time only. Use the second form when the situation flight characteristics for the named collision warning situation are specified by range only. Use the third form when the situation flight characteristics for the named collision warning situation are specified by both time and range. In all cases, the values for *CWS_Def.Time.Min*, *CWS_Def.Time.Max*, *CWS_Def.Range.Min*, and *CWS_Def.Range.Max* must have exactly one digit after the decimal point (e.g., 44.0).

Note 11: Repeat the quadruple (cws_name, severity, predicate, partition) once for every collision warning situation you have defined in Step 1. Each ordered quadruple is enclosed in parentheses and has a trailing comma except for the last ordered quadruple in the list. Furthermore, each quadruple must be ordered in decreasing severity level.

- **PT_.TRF and PT.TRF**—To adapt pt_.trf for Potential_Threat, you create a metafile called pt_.meta which contains the following statements:

  {^module!include(pt_spec. "pt_.trf")}
  {^pt_spec.pt!spec(
            ((cws_name : {*CWS_Name*},
                      severity : {*Severity*},
                      predicate : see note 12,
                      partition : {*CWS.Partition*},
                      alarm : {*Response.Alarm* [see note 13]},
                      atc_msg : {*Response.ATC_Msg* [see note 13]},
                      inter_air_msg : {*Response.Inter_Air_Msg* [see note 13]},
                      corrective : {*Response.Corrective_Msg* [see note 13]}).
                 . . . see note 14
)
)
  }

To adapt pt.trf for Potential_Threat, you create a metafile called pt.meta which contains the following statements:

  {^module!include(pt_body. "pt.trf")}
  {^pt_body.pt!body(
            ((cws_name : {*CWS_Name*},
                      severity : {*Severity*},
                      predicate : see note 12,
                      partition : {*CWS.Partition*},
                      alarm : {*Response.Alarm* [see note 13]}.
                      atc_msg : {*Response.ATC_Msg* [see note 13]},
                      inter_air_msg : {*Response.Inter_Air_Msg* [see note 13]}.
                      corrective : {*Response.Corrective_Msg* [see note 13]}).
                 . . . see note 14
)
)
  }

Note 12:   The predicate component of the ordered tuple (cws_name, severity, predicate, partition, alarm, atc_msg, inter_air_msg, corrective) has one of the following forms depending on the situation flight characteristics for the named collision warning situation.

  (time : (min : {*CWS_Def.Time.Min*}, max : {*CWS_Def.Time.Max*}))

  (range : (min : {*CWS_Def.Range.Min*}, max: {*CWS_Def.Range.Max*}))

  (t_and_r : (t_min : {*CWS_Def.Time.Min*},
             t_max: {*CWS_Def.Time_Max*},
             r_min: {*CWS_Def.Range.Min*},
             r_max: {*CWS_Def.Range.Max*}))

Use the first form when the situation flight characteristics for the named collision warning situation are specified by time only. Use the second form when the situation flight characteristics for the named collision warning situation are specified by range only. Use the third form when the situation flight characteristics for the named collision warning situation are specified by both time and range. In all cases, the values for *CWS_Def.Time.Min*, *CWS_Def.Time.Max*, *CWS_Def.Range.Min*, and *CWS_Def.Range.Max* must have exactly one digit after the decimal point (e.g., 44.0).

Note 13: The value must be either True or False (using the capitalization as indicated).

Note 14: Repeat the tuple (cws_name, severity, predicate, partition, alarm, atc_msg. inter_air_msg, corrective) once for every collision warning situation you have defined in Step 1. Each ordered tuple is enclosed in parentheses and has a trailing comma except for the last tuple in the list. Furthermore, you must order each tuple in decreasing severity level.

- **RTPB_.TRF and RTPB.TRF**—To adapt tdb_.trf for Radar_Target_Priority_Buffer, you create a metafile called rtpb_.meta which contains the following statements:

```
{ ^ module!include(tdb_spec, "tdb_.trf")}
{ ^ tdb_spec.tdb!spec(
        name : {Radar_Target_Priority_Buffer}.
        length : {20},
        message : ( module_name : {Potential_Threat}.
                    data_type : {pt_handle}).
        consumer : (    (consumer_name : {CWS_Name}, priority : {Severity}).
                        . . . see note 15
                    )
    )
}
```

To adapt tdb.trf for Radar_Target_Priority_Buffer, you create a metafile called rtpb.meta which contains the following statements:

```
{ ^ module!include(tdb_body, "tdb.trf")}
{ ^ tdb_body.tdb!body(
        name : {Radar_Target_Priority_Buffer},
        length : {20}.
        message : ( module_name : {Potential_Threat},
                    data_type : {pt_handle}).
        consumer : (    (consumer_name : {CWS_Name}, priority : {Severity}).
                        . . . see note 15
                    )
    )
}
```

Note 15: Repeat the pair (consumer_name, priority) once for every collision warning situation you have defined in Step 1. Each pair is enclosed in parentheses and has a trailing comma except for the last ordered pair in the list. Furthermore, you must order the pairs in decreasing priority value.

- **TB_.TRF and TB.TRF**—To adapt tdb_.trf for Target_Buffer, you c ate a metafile called tb_.meta which contains the the following statements:

```
{ ^ module!include(tdb_spec, "tdb_.trf")}
{ ^ tdb_spec.tdb!spec(
          name : {Target_Buffer},
          length : {20},
          message : ( mod le_name : {Potential_Threat},
                      data_type : {target_info}),
          consumer : ()
  )
}
```

To adapt tdb.trf for Target_Buffer, you create a metafile called tb.meta which contains the following statements:

```
{ ^ module!include(tdb_body, "tdb.trf")}
{ ^ tdb_body.tdb!body(
          name : {Target_Buffer},
          length : {20},
          message : ( module_name : {Potential_Threat},
                      data_type : {target_info}),
          consumer : ()
  )
}
```

Step 4. Compose the Components

You compose the adapted code components into an executable ATD/CWM system by copying the files from the Apollo system to the target system, compiling the Ada files, compiling the one C file, and linking them to form an executable. These steps are described in greater detail below. The target system is defined as VAX/VMS V5.4-2. This target system must also have VAX Ada V2.2-38 and VAX C V3.2-044 on it.

Move the files to the target

You must move the Ada and C files (i.e., files with the ".a" or ".c" suffix) over to the target system prior to starting the compilations. If the only Ada and C files that reside in the local directory containing the adapted code components are ATD/CWM Ada and C files, the following command will copy all Ada files over to the target machine:

rcp *.a *vax_host:'[home_dir...]'*

The syntax for the target machine and directory is **host:path** where **host** is the name of the remote target machine and **path** is a single quoted pathname on the target machine. If needed, a **csh** script has been created that will only copy the exact ATD/CWM Ada files over to the target system. This copy script, entitled **transport.csh**, is located in the following Apollo directory:

//venus/local/public/atd_cwm_adaptable_components/code_components

This copy script accepts one command line parameter which is the remote node and directory where the Ada files are to be copied. The syntax for the command line parameter is identical to the one described above for the **rcp** command. You will need to edit this copy script to remove any Ada files that are not to be copied (i.e., those components not selected in Step 2). An example usage of this copy script, assuming a VAX target, is illustrated as follows:

transport.csh < vax_host > :'[< home_dir > . < some_subdir >]'

Compile the Ada files

It is assumed that you have some knowledge of how to establish and use a suitable Ada environment on VAX/VMS and that you have some knowledge of the VAX/VMS command language before you attempt this step. It is assumed that all necessary adaptations have been performed in creating the needed Ada files prior to attempting compilation.

Each adapted Ada code component, each Ada code components that did not require any adaptation, and those text files created to instantiate Ada generics must be compiled in a specific compilation order. The following list reflects the proper compilation order where components enclosed by square brackets ([ ]) are compiled only if they were selected in Step 2 and italicized "names" are substituted with the names you selected for those components:

pq_.a
na_.a
am_.a
atdd_.a
file containing adapted pt_.trf for Potential_Threat
sd_.a
cwss_.a
atd_.a
atc_.a
nav_.a
[*file containing adapted cd_.trf for Communication_Device*]
[*file containing adapted tdb_.trf for Communication_Buffer*]
[*file containing adapted comm_.trf for Communication*]
file containing adapted tdb_.trf for Radar_Target_Priority_Buffer
file containing adapted tdb_.trf for Target_Buffer
ha_.a
radar_.a
am.a
file containing text to adapt Aircraft_Motion
atc.a
atd.a
atdd.a
[*file containing adapted cd.trf for Communication_Device*]
[*file containing adapted comm.trf for Communication*]
[*file containing adapted tdb.trf for Communication_Buffer*]
file containing adapted cwss.trf for Collision_Warning_Situation_Status
ha.a

it.a
na.a
nav.a
pq.a
file containing adapted pt.trf for Potential_Threat
radar.a
file containing adapted tdb.trf for Radar_Target_Priority_Buffer
sd.a
file containing adapted tdb.trf for Target_Buffer

A predefined VAX compilation script, entitled **vax_compile.com**, reflects this compilation order and is available for compiling these Ada files. This compilation script is available in the following Apollo directory and should be copied over to the VAX:

//venus/local/public/atd_cwm_adaptable_components/code_components

If the adapted Ada components were created using the predefined Ada filenames contained in the **csh** script, **adapt.csh**, you will only need to edit out any Ada files from this compile script that were not copied over to the VAX (i.e., because they were not selected in Step 2). However, if the predefined names were not used, you will need to substitute the new names for the italicized items as described in the above list.

The VAX/VMS Ada compiler generates a series of extraneous warning messages for the following components:

atd.a
atdd.a
file containing adapted cwss.trf for Collision_Warning_Situation_Status
file containing adapted pt.trf for Potential_Threat
radar.a
sd.a

You can ignore these. Any other messages should be reported to the ATD/CWM domain engineers.

Compile the C file

It is assumed that you have some knowledge of how to use the VAX/VMS C compiler on VAX/VMS and that you have some knowledge of the VAX/VMS command language before you attempt this step. Before compiling component xlibrary.c, you must define a VAX/VMS logical name X11 to be the name of the VAX/VMS directory that contains the C include file X.h. This equivalence name is defined on the VAX as follows:

$ define X11 decw$include

Once you have defined this logical name, you can compile xlibrary.c using the C compiler. Any messages reported by the C compiler should be reported to the ATD/CWM domain engineers.

Creating an executable

Once you have compiled these components, they are linked together to form the desired executable ATD/CWM system. The entry point for your system is named Atd_Cwm. You must also link in the following VAX/VMS libraries:

- decw$xlibshr.exe

- vaxcrtl.olb

A linking script file, entitled **link.com**, can be found in the following Apollo directory:

> //venus/local/public/atd_cwm_adaptable_components/code_components

The linker generates a series of extraneous warning messages for components.

> atd.a
> atdd.a
> *file containing adapted cwss.trf for Collision_Warning_Situation_Status*
> *file containing adapted pt.trf for Potential_Threat*
> radar.a
> sd.a

These are caused by the extraneous messages generated by the VAX/VMS Ada compiler. You can ignore these. Any other messages should be reported to the *ATD/CWM domain engineers*.

Step 5. Executing Your System

Your ATD/CWM system is intended to execute on VAX/VMS using your Apollo terminal as an X-terminal client. Before you can execute your ATD/CWM system, you must perform the following steps:

- Type the following command at the VAX/VMS DCL prompt:

    - set disp/create/node = *XX*/trans = wintcp

  where *XX* is the name of the Apollo node that you wish to use as an X-terminal.

- On the Apollo node called *XX*, ensure that the X-server is running. It must be running before you can perform the next step.

- Type the following command at the Apollo prompt on the Apollo node called *XX*:

    - /usr/bin/X11/xhost +

You can now start your ATD/CWM system on the VAX by issuing the following command:

    - $ run/nodebug atd_cwm

Your ATD/CWM system will execute using simulated radar and ATC data.

This page intentionally left blank.

8. ATD/CWM PROCESS SUPPORT

1. APPLICATION ENGINEERING USER'S GUIDE

The application engineering process for the ATD/CWM domain consists of two activities: Application Modeling and Application Production. You must complete the Application Modeling activity before the Application Production activity is started.

1. APPLICATION MODELING

Application Modeling consists of two activities: specification and validation. Specification analyzes a customer's statement of needs to produce an Application Model. The Application Model expresses requirements and engineering decisions that describe an instance of a family of systems intended to satisfy those needs.

In this activity, you will be describing an ATD_CWM system which when installed in an aircraft will monitor air traffic in a surveillance area and detect collision warning situations. This system will monitor flight characteristics (e.g., altitude, bearing, range) of potential threats and show the flight characteristics on a display within the host aircraft's cockpit. The ATD/CWM system obtains flight characteristics from either messages transmitted by potential threats or air traffic control centers. This system will also detect collision warning situations and take appropriate actions such as displaying collision warning characteristics and corrective action advisory messages on a display within the host aircraft's cockpit, transmitting inter–air messages to potential threats, and transmitting advisory messages to an air traffic control center.

The following sections describe the sequence of steps you, the application engineer, perform to develop an ATD/CWM application.

1.1. Specification

You must specify the ATD/CWM Application Model before generation activities can be done. The steps that you follow are listed below. The forms you will need to fill in are provided with each step. Each form has the following organization.

Decision	Mnemonic	Value

The first column identifies the decisions (i.e., the requirements variations); the second column contains a mnemonic which is a shorthand identifier for the decision; and the third column records your decisions. You can repeat the form pertaining to decisions you must make for collision warning situations as often as necessary. This allows you to capture your decisions for each collision warning situation. The steps describe what decisions you must make and permissible values for each decision.

1. Define the Application Model name. This name is a case-sensitive alphanumeric text string. The length of this string must be between 1 and 64 characters. You enter this name in the form shown in Table 8-1.

Table 8-1. Application Model Name

Decision	Mnemonic	Value
Application Model Name	Model_Name	

2. You can perform the following steps in any order. However, you must perform all of them before you have completed an Application Model.

   2.1. Define the host aircraft characteristics. There are three decisions that you must make: the surveillance area radius, the icon shape for the host aircraft, and the ATC_Msg message format. You enter your decisions for these requirements variations in the form shown in Table 8-2 opposite labels marked "Surveillance_Area," "Host_Aircraft_Shape," and "ATC_Message_Mode," respectively. A brief description of these decisions and their associated value space follows.

   Surveillance_Area   Radius (in nautical miles) of the surveillance area that the ATD/CWM monitors. The surveillance area is a sphere whose origin is the host aircraft's position. You must chose an integer in the range 10 to 300, inclusive.

   Host_Aircraft_Shape   Icon shape for the host aircraft when it is displayed. You can select only one of **circle**, **square**, or **triangle**.

   ATC_Message_Mode   Designates the format for the ATC_Msg messages sent from the host aircraft to an air traffic control center. You can select only one of the following values:
   A – Transponder code only
   C – Transponder code plus altitude

Table 8-2. Host Aircraft Characteristics

Decision	Mnemonic	Value
Host_Aircraft Characteristics	————————————	————————————
Surveillance Area	Surveillance_Area	
Host Aircraft Shape	Host_Aircraft_Shape	
ATC_Message_Mode	Message_Mode	

   You only need to select a value for "ATC_Message_Mode" when you have at least one collision warning situation response which has the "Response to ATC" decision marked true.

   2.2. Define potential threat characteristics. These are the characteristics unique to potential threats. There are three decisions that you must make: the criteria for distinguishing between "identified" and "unidentified" aircraft, the icon shape for "identified" aircraft, and

the icon shape for "unidentified" aircraft. You enter your decisions for these requirements variations in the form shown in Table 8-3 opposite labels marked "Identification Requirements," "Shape of Identified Aircraft," and "Shape of Unidentified Aircraft," respectively. A brief description of these decisions and their permitted value space follows.

Identification Requirements   A set that defines criteria for a potential threat to be considered "identified." You can select one or more of **airspeed** or **altitude**. To select element, say airspeed, means that the value for airspeed must be known for a potential threat to be designated as "identified." Selecting both elements means that both values must be known.

Shape of Identified Aircraft   Icon shape for an "identified" potential threat when it is displayed. You can select only one of **circle, square,** or **triangle**.

Shape of Unidentified Aircraft

    Icon shape for an "unidentified" potential threat when it is displayed. You can select only one of **circle, square,** or **triangle**.

Table 8-3. Potential Threat Characteristics

Decision	Mnemonic	Value
Potential Threat Characteristics	————————————	————————————
Identification Requirements	ID_Req	
Shape of Identified Aircraft	ID_Shape	
Shape of Unidentified Aircraft	UID_Shape	

2.3. Define Collision Warning Situation. You enter the name of a specific collision warning situation in the form shown in Table 8-4 opposite label "Collision Warning Situation Name." This name is a case-insensitive alphanumeric identifier (no spaces allowed) having a maximum length of 64 characters. The name "normal" is reserved and cannot be specified by the application engineer, including all upper- and lower-case variations. Steps 2.3, 2.3.1, 2.3.2, 2.3.3, and 2.3.4 are repeated as often as there are collision warning situations to specify. You will need to complete Table 8-4 (shown on Page 8-8) once for every collision warning situation in your ATD/CWM system.

A collision warning situation consists of three portions: the defining characteristics, the appropriate response performed by the system, and the desired display characteristics. You must specify these for each collision warning situation using the following steps.

2.3.1.   Define the collision warning situation characteristics. There are two required decisions: the aircraft partition to which this situation applies and the severity of this collision warning situation. You enter your decisions for these requirements variations in the form shown in Table 8-4 opposite labels marked "Situation Aircraft Partition" and "Situation Severity," respectively. A brief description of these decisions and their permitted value space follows.

Situation Aircraft Partition	Indicates the potential threat partition for which this collision warning situation applies. You can select only one of **ID**, **UID**, or **ALL**. ID is "identified"; UID is "unidentified"; ALL is both.
Situation Severity	Relative probability that a collision is likely to occur. The higher the severity, the more likely a collision will occur. By definition, the predefined normal situation has the lowest severity. You can select a severity level in the range 0.00 to 1.00 having a resolution of 0.01.

There are also four optional decisions: the minimum and maximum allowed time before the flight paths of a potential threat and the host aircraft intersect, and the minimum and maximum distance a potential threat is from the host aircraft. You can specify the minimum and maximum time or the minimum and maximum range or both. However, you must specify at least one of these pairs. If both are specified, it is interpreted as a logical "OR" (i.e., time OR range). You enter your decision for these requirements variations in the form shown in Table 8-4 opposite labels marked "Time_Min," "Time_Max," "Range_Min," and "Range_Max," respectively. If airspeed is unknown, then 1000 nautical miles per hour is assumed. A brief description of these decisions and their permitted value space follows.

Time Minimum	Minimum allowed elapsed time before the flight path's of the potential threat and host aircraft intersect. You can select a minimum time value in the range 1 to 300 seconds.
Time Maximum	Upper bound on the allowed elapsed time before the flight paths of the potential threat and host aircraft intersect. You can select a maximum time value in the range 1 to 300 seconds.
Range Minimum	Minimum distance the potential threat is from the host aircraft. The upper limit is determined by the value chosen for the Surveillance_Area decision. You can select a minimum range value in the range 0 to X nautical miles, inclusive. X denotes the value chosen for the Surveillance_Area decision.
Range Maximum	Upper bound on the potential threat is from the host aircraft. The upper limit is determined by the value chosen for the Surveillance_Area decision. You can select a maximum range value in the range 0 to X nautical miles, inclusive. X denotes the value chosen for the Surveillance_Area decision.

2.3.2. Define the situation response. There are five decisions that you must make: should the ATD/CWM system send notification to the air traffic control center; should the system send notification to the intruding potential threat; should the system

display a corrective action message; should the system ring the audible alarm; and what transponder code the ATD/CWM should use in the ATC_Msg or Inter_Air_Msg. You enter your decisions for these requirements variations in the form shown in Table 8-4 opposite labels marked "Response to ATC," "Response to other Aircraft," "Corrective Action Response," "Alarm," and "Code," respectively. A brief description of these decisions and their permitted value space follows.

Response to ATC

Designates whether a message is sent to the nearest air traffic control center. You can select either **True** or **False**. **True** means to send the message; **false** means do not send the message.

Response to other Aircraft

Designates whether a message is sent to the appropriate potential threat. You can select either **True** or **False**. **True** means to send the message; **false** means do not send the message.

Corrective Action Response

Designates whether a corrective action advisory message is displayed on the ATD. You can select either **True** or **False**. **True** means the message is displayed; **false** means that the message is not displayed.

Alarm

Designates whether the audible alarm should be rung when a potential threat migrates into this collision warning situation from a lower severity. You can select either **True** or **False**. A value of **false** means to not ring the alarm. A value of **true** means to ring the alarm. In this case, a pitch and duration must be specified as well.

Code

Designates the four-digit transponder code to use in the ATC_Msg and Inter_Air_Msg. You must chose a 4-digit integer in the range 0000 to 7777, inclusive. excluding the following reserved codes:

7500
7600 through 7677, inclusive
7700 through 7777, inclusive

The last two digits of your code must always read 00. Furthermore, each digit is restricted to values in the range 0 to 7, inclusive. You do not need to make a selection for this decision when your choices for "Response to ATC" and "Response to other Aircraft" decisions are both false.

If "Response to ATC" is marked **true**, you must chose a value for the "ATC_Message_Mode" decision found in Table 8-2.

2.3.3. If you chose **true** value for the "Alarm" decision, you must specify the alarm characteristics by specifying a pitch and a duration for the audible alarm. You enter your decisions for these requirements variations in the form shown in Table 8-4 opposite labels marked "Pitch" and "Duration," respectively. A brief description of these decisions and their permitted value space follows.

Pitch   What frequency, in hertz, at which the audible alarm is rung. You can select an integer-valued frequency in the range 1,000 to 10,000, inclusive.

Duration   How long to ring the audible alarm. You can select a time duration in the range 0.01 to 10.0 seconds, inclusive, having a resolution of 0.01 seconds.

2.3.4. Define the situation display characteristics. There are seven decisions that you must make: icon color of host aircraft, icon color of identified potential threats, whether the icons for identified potential threats should blink, whether the icons for identified potential threats should be filled in, icon color of unidentified potential threats, whether the icon for unidentified potential threats should blink, and whether the icons for unidentified potential threats should be filled in. You enter your decisions for these requirements variations in the form shown in Table 8-4 opposite labels marked "Color of Host Aircraft," "Color of Identified Potential Threats," "Blinking Identified Potential Threats," "Fill Identified Potential Threats," "Color of Unidentified Potential Threats," "Blinking Unidentified Potential Threats," and "Fill Unidentified Potential Threats," respectively. A brief description of these decisions and their permitted value space follows.

Color of Host Aircraft   Icon color for the host aircraft. You can select only one color: **red, yellow, pink, orange, blue, green, white, black, purple, indigo,** or **violet.**

Color of Identified Potential Threat
                                         Icon color for the identified potential threat. You can select only one color: **red, yellow, pink, orange, blue, green, white, black, purple, indigo,** or **violet.**

Blinking Identified Potential Threat
                                         You can select either **True** or **False. True** means that the icon for the identified potential threat should blink in this collision warning situation. **False** means that it should not blink.

Fill Identified Potential Threat
                                         You can select either **True** or **False. True** means that the icon for the identified potential threat should be filled in (i.e., color the icon interior). **False** means do not fill the icon.

Color of Unidentified Potential Threat
                                         Icon color for the unidentified potential threat. You

can select only one color: red, **yellow, pink, orange, blue, green, white, black, purple, indigo,** or **violet.**

Blinking Unidentified Potential Threat

You can select either **True** or **False. True** means that the icon for the unidentified potential threat should blink in this collision warning situation. **False** means that it should not blink.

Fill Unidentified Potential Threat

You can select either **True** or **False. True** means that the icon for the unidentified potential threat should be filled in (i.e., color the icon interior). **False** means do not fill the icon.

Table 8-4. Collision Warning Situation

Decision	Mnemonic	Value
Collision Warning Situation	——	——
Collision Warning Situation Name	CWS_Name	
Situation Definition	——	——
Situation Aircraft Partition	Partition	
Situation Severity	Severity	
Situation Flight Characteristics	——	——
Time	——	——
Min	Time_Min	
Max	Time_Max	
Range	——	——
Min	Range_Min	
Max	Range_Max	
Situation Response	——	——
Response to ATC	ATC_Msg	
Response to other Aircraft	Inter_Air_Msg	
Corrective Action Response	Corrective_Msg	
Alarm	Alarm	
Code	Code	_ _ 0 0
Alarm Characteristics	——	——
Pitch	Alarm_Pitch	
Duration	Alarm_Duration	
Situation Display	——	——
Color of Host Aircraft	Host_Color	
Color of Identified Potential Threats	ID_Color	
Blinking Identified Potential Threats	ID_Blink	
Fill Identified Potential Threats	ID_Fill	
Color of Unidentified Potential Threats	UID_Color	
Blinking Unidentified Potential Threats	UID_Blink	
Fill Unidentified Potential Threats	UID_Fill	

1.2. VALIDATION

Having completed the specification, you can validate the Application Model by performing the following checks shown below. All checks must pass to have a validated Application Model. If any of the checks fail, you must correct the necessary portions of the Application Model and subsequently validate it again.

1. Your Application Model contains at least one collision warning situation.

2. Every collision warning situation contains values for all required fields.

3. You have marked the "Corrective Action Response" decision **true** for at least one collision warning situation.

4. You have specified a value for the ATC_Message_Mode if, and only if, there is at least one collision warning situation response which has the "Response to ATC" decision marked **true**.

5. For each collision warning situation in which Time is specified, the minimum time value ("Time_Min") must be less than or equal to the maximum time ("Time_Max").

6. For each collision warning situation in which Range is specified, the following checks are done.

   a) The minimum range value ("Range_Min") must be less than or equal to the maximum range ("Range_Max").

   b) The minimum range value ("Range_Min") must be less than or equal to the Surveillance_Area range specified in your Application Model and greater than or equal to zero.

   c) The maximum range value ("Range_Max") must be less than or equal to the Surveillance_Area range specified in your Application Model and greater than or equal to zero.

7. You have specified mutually exclusive icon shapes for the host aircraft, identified potential threats, and unidentified potential threats.

8. The set of collision warning situations do not overlap. It is likely that all of the collision warning situations will be specified either in terms of time or in terms of range but not a mixture of both. Thus, these time-based or range-based situations should not overlap.

2. APPLICATION PRODUCTION

You must have successfully validated your Application Model before you can generate the Application Products. If your Application Model has been validated, you follow the following steps to produce the desired application.

   1. Application Model Transformation.

   You must transform your validated ATD/CWM Application Model (from its external form) into an equivalent internal form expressed in terms of the ATD/CWM Decision Model before you proceed with the remaining activities of the Generation Procedure. To do this transformation, you fill in forms that correspond to decision classes in the ATD/CWM Decision Model. You derive the values for these forms from your ATD/CWM Application Model. Page 7-153 describes how to do this step in more detail.

   2. Select the Adaptable Components.

   You use the information you captured in the preceding step to select adaptable components. You are provided a group of adaptable components and selection criteria for each component. To select the adaptable component, you must evaluate the selection criteria for each adaptable component. Page 7-157 describes in more detail how you perform this step.

3. Adapt the Components.

    Each of the adaptable code components is normally implemented by two parts: a specification and a body. You adapt either the specification, body, or both for a given adaptable code component. You adapt only those components you selected in the preceding step. Page 7-160 describes in greater detail how you adapt the components.

4. Compose the Components.

    You compose the adapted code components into an executable ATD/CWM system by compiling the source code files and linking them to form an executable. The basic steps are moving the components to the target system (i.e., the system on which the ATD/CWM system executes), compiling the Ada and C code components, and finally linking the compiled code components together to form your ATD/CWM system. Page 7-168 describes in greater detail how you compose the selected components for your ATD/CWM system.

Once you have successfully built the desired application, you can execute your ATD/CWM system on the target hardware by following the steps described on page 7-171 .

3. RUNTIME VALIDATION

You and your customer can evaluate your ATD/CWM system by performing the following checks as the system executes.

1. The ATD/CWM system recognizes each collision warning situation.

2. The ATD/CWM system performs the desired actions in response to detected collision warning situations.

3. Each aircraft in the surveillance area is displayed with the desired identifying icon.

9. ATD/CWM APPLICATION MODEL

NOTE: The requirements for the ATD/CWM system captured in the following Application Model are shown in Appendix D. This Application Model was developed by manually following the ATD/CWM Application Engineering User's Guide contained in the ATD/CWM Process Support work product (Section 8).

Decision	Mnemonic	Value
Application Model Name	Model_Name	ATD/CWM System_1

Decision	Mnemonic	Value
Host_Aircraft Characteristics	—	—
Surveillance Area	Surveillance_Area	125
Host Aircraft Shape	Host_Aircraft_Shape	circle
ATC_Message_Mode	Message_Mode	A

Decision	Mnemonic	Value
Potential Threat Characteristics	—	—
Identification Requirements	ID_Req	(airspeed)
Shape of Identified Aircraft	ID_Shape	triangle
Shape of Unidentified Aircraft	UID_Shape	square

Decision	Mnemonic	Value
Collision Warning Situation	—	—
Collision Warning Situation Name	CWS_Name	Monitored
Situation Definition	—	—
Situation Aircraft Partition	Partition	ALL
Situation Severity	Severity	0.10
Situation Flight Characteristics	—	—
Time	—	—
Min	Time_Min	
Max	Time_Max	
Range	—	—
Min	Range_Min	0
Max	Range_Max	125

Decision	Mnemonic	Value
Situation Response	————————	————————————
Response to ATC	ATC_Msg	false
Response to other Aircraft	Inter_Air_Msg	false
Corrective Action Response	Corrective_Msg	false
Alarm	Alarm	false
Code	Code	7400
Alarm Characteristics	————————	————————————
Pitch	Alarm_Pitch	
Duration	Alarm_Duration	
Situation Display	————————	————————————
Color of Host Aircraft	Host_Color	white
Color of Identified Potential Threats	ID_Color	orange
Blinking Identified Potential Threats	ID_Blink	false
Fill Identified Potential Threats	ID_Fill	false
Color of Unidentified Potential Threats	UID_Color	green
Blinking Unidentified Potential Threats	UID_Blink	false
Fill Unidentified Potential Threats	UID_Fill	false

Decision	Mnemonic	Value
Collision Warning Situation	————————	————————————
Collision Warning Situation Name	CWS_Name	Possible
Situation Definition	————————	————————————
Situation Aircraft Partition	Partition	ALL
Situation Severity	Severity	0.20
Situation Flight Characteristics	————————	————————————
Time	————————	————————————
Min	Time_Min	60
Max	Time_Max	90
Range	————————	————————————
Min	Range_Min	
Max	Range_Max	
Situation Response	————————	————————————
Response to ATC	ATC_Msg	true
Response to other Aircraft	Inter_Air_Msg	false
Corrective Action Response	Corrective_Msg	false
Alarm	Alarm	true
Code	Code	7200
Alarm Characteristics	————————	————————————
Pitch	Alarm_Pitch	2500

Decision	Mnemonic	Value
Duration	Alarm_Duration	5.00
Situation Display	--------------	----------------------
Color of Host Aircraft	Host_Color	white
Color of Identified Potential Threats	ID_Color	orange
Blinking Identified Potential Threats	ID_Blink	false
Fill Identified Potential Threats	ID_Fill	false
Color of Unidentified Potential Threats	UID_Color	green
Blinking Unidentified Potential Threats	UID_Blink	false
Fill Unidentified Potential Threats	UID_Fill	false

Decision	Mnemonic	Value
Collision Warning Situation	--------------	----------------------
Collision Warning Situation Name	CWS_Name	Potential
Situation Definition	--------------	----------------------
Situation Aircraft Partition	Partition	ALL
Situation Severity	Severity	0.30
Situation Flight Characteristics	--------------	----------------------
Time	--------------	----------------------
Min	Time_Min	30
Max	Time_Max	60
Range	--------------	----------------------
Min	Range_Min	
Max	Range_Max	
Situation Response	--------------	----------------------
Response to ATC	ATC_Msg	true
Response to other Aircraft	Inter_Air_Msg	true
Corrective Action Response	Corrective_Msg	true
Alarm	Alarm	true
Code	Code	7300
Alarm Characteristics	--------------	----------------------
Pitch	Alarm_Pitch	4000
Duration	Alarm_Duration	5.00
Situation Display	--------------	----------------------
Color of Host Aircraft	Host_Color	white
Color of Identified Potential Threats	ID_Color	pink
Blinking Identified Potential Threats	ID_Blink	true
Fill Identified Potential Threats	ID_Fill	true
Color of Unidentified Potential Threats	UID_Color	blue

Decision	Mnemonic	Value
Blinking Unidentified Potential Threats	UID_Blink	true
Fill Unidentified Potential Threats	UID_Fill	true

Decision	Mnemonic	Value
Collision Warning Situation	——————	——————————
Collision Warning Situation Name	CWS_Name	Imminent
Situation Definition	——————	——————————
Situation Aircraft Partition	Partition	ALL
Situation Severity	Severity	0.50
Situation Flight Characteristics	——————	——————————
Time	——————	——————————
Min	Time_Min	0
Max	Time_Max	30
Range	——————	——————————
Min	Range_Min	
Max	Range_Max	
Situation Response	——————	——————————
Response to ATC	ATC_Msg	true
Response to other Aircraft	Inter_Air_Msg	true
Corrective Action Response	Corrective_Msg	true
Alarm	Alarm	true
Code	Code	7100
Alarm Characteristics	——————	——————————
Pitch	Alarm_Pitch	8000
Duration	Alarm_Duration	5.00
Situation Display	——————	——————————
Color of Host Aircraft	Host_Color	red
Color of Identified Potential Threats	ID_Color	yellow
Blinking Identified Potential Threats	ID_Blink	true
Fill Identified Potential Threats	ID_Fill	true
Color of Unidentified Potential Threats	UID_Color	purple
Blinking Unidentified Potential Threats	UID_Blink	true
Fill Unidentified Potential Threats	UID_Fill	true

10. ATD/CWM APPLICATION SOFTWARE

Note: The components shown in this section were produced by manually following the Application Production portion of the ATD/CWM Application User's Guide (Section 8) for the Application Model shown on page 9-1. For clarity and illustrative purposes, only some of the code components and documentation components are shown below.

Code Components

1. Audible_Alarm

Spec

```
--
-- Audible_Alarm (AA)
--
-- This module determines the frequency and duration at which
-- to ring the audible alarm for a specified collision warning
-- situation.
--
with Potential_Threat;
package Audible_Alarm is

    procedure ring_alarm(cws : in Potential_Threat.cws_id);

end Audible_Alarm;
```

Body

```
--
-- Audible_Alarm (AA) body
--
-- The audible_alarm device generates a tone that can be heard
-- within the host_aircraft cockpit.
--
with Potential_Threat;
with Audible_Alarm_Device;
package body Audible_Alarm is

    procedure ring_alarm(cws : in Potential_Threat.cws_id)
    is
    begin
       case cws is
          when Potential_Threat.Possible =>
             Audible_Alarm_Device.ring_alarm(f => 2500, d => 5.00);
```

```
                  when Potential_Threat.Potential =>
                     Audible_Alarm_Device.ring_alarm(f => 4000, d => 5.00);
                  when Potential_Threat.Imminent =>
                     Audible_Alarm_Device.ring_alarm(f => 8000, d => 5.00);
                  when others =>
                     return;
            end case;
         end ring_alarm;
      end Audible_Alarm;
```

2. Collision_Warning_Situation_Status

Spec

```
--
-- Collision Warning Situation Status (CWSS) spec
--
-- This module determines the collision warning situation status
-- for the given potential threat and host aircraft.
--
with Potential_Threat;
package Collision_Warning_Situation_Status is

   function determine_cws_status(threat : in Potential_Threat.pt_handle)
                                            return Potential_Threat.cws_id;

   function determine_host_status return Potential_Threat.cws_id;

end Collision_Warning_Situation_Status;
```

Body (excerpt)

```
--
-- Collision Warning Situation Status (CWSS)  package body
--
-- This module determines the collision warning situation status
-- for the given potential_threat and host_aircraft.
--
with Potential_Threat;

with Physical_Quantities; use Physical_Quantities;

with Situation_Dynamics;
with Text_IO;

package body Collision_Warning_Situation_Status is

--
-- This routine keeps track of the number of potential
-- threats in each collision situation. This enables us to
-- quickly determine the host aircraft status when
-- requested to provide it.
--
   target_count : array(Potential_Threat.cws_id'first ..
                        Potential_Threat.cws_id'last) of integer := (others
   => 0);
```

```
--
-- Determine the collision warning situation status of the specified
-- potential threat.
--
    function determine_cws_status(threat : in Potential_Threat.pt_handle)
                                        return Potential_Threat.cws_id
    is
        airspeed_and_altitude_valid : boolean;

        time_to_intersect : Physical_Quantities.seconds;

        target_range : Physical_Quantities.nautical_mile;

        old_cws_status, new_cws_status : Potential_Threat.cws_id;
    begin
        airspeed_and_altitude_valid := Potential_Threat.altitude_valid(threat)
and then

Potential_Threat.velocity_valid(threat);

        target_range := Potential_Threat.get_range(threat);

        if (airspeed_and_altitude_valid) then
            time_to_intersect := Situation_Dynamics.get_elapsed_time(threat);
        end if;

        old_cws_status := Potential_Threat.get_cws_status(threat);
        if (              .

            (airspeed_and_altitude_valid) and then
            (0.0 <= time_to_intersect and then
                        time_to_intersect < 30.0)) then
                new_cws_status := Potential_Threat.Imminent;

        elsif (

            (airspeed_and_altitude_valid) and then
            (30.0 <= time_to_intersect and then
                        time_to_intersect < 60.0)) then
                new_cws_status := Potential_Threat.Potential;

        elsif (

            (airspeed_and_altitude_valid) and then
            (60.0 <= time_to_intersect and then
                        time_to_intersect < 90.0)) then
                new_cws_status := Potential_Threat.Possible;

        elsif (

            (0.0 <= target_range and then target_range < 125.0)) then
                new_cws_status := Potential_Threat.Monitor;

        else
            new_cws_status := Potential_Threat.normal;
```

```
                  end if;
                  if (target_count(old_cws_status) /= 0) then
                     target_count(old_cws_status) := target_count(old_cws_status) - 1;
                  end if;
                  target_count(new_cws_status) := target_count(new_cws_status) + 1;
                  return new_cws_status;
            exception
               when constraint_error => text_io.put_line("determine cws CE"); return
         Potential_Threat.normal;
               when numeric_error => text_io.put_line("determine cws NE"); return
         Potential_Threat.normal;
               when others => text_io.put_line("determine cws Bozo error"); return
         Potential_Threat.normal;
                  end determine_cws_status;


         --
         -- Determine the collision warning situation status of
         -- the host aircraft. Each the number of potential threats
         -- in each situation category starting with the most severe
         -- situation and progressing to the least severe. The
         -- first collision warning situation encountered which has
         -- a non-zero target count is the status of the host aircraft.
         -- If all situations have zero potential threats, then the
         -- status of the host aircraft is "normal".
         --
            function determine_host_cws_status return Potential_Threat.cws_id
            is
            begin
               if (

         target_count(Potential_Threat.Imminent) /= 0) then
               return Potential_Threat.Imminent;

               elsif (

         target_count(Potential_Threat.Potential) /= 0) then
               return Potential_Threat.Potential;

               elsif (

         target_count(Potential_Threat.Possible) /= 0) then
               return Potential_Threat.Possible;

               elsif (

         target_count(Potential_Threat.Monitor) /= 0) then
               return Potential_Threat.Monitor;

               else
                  return Potential_Threat.normal;
               end if;
            end determine_host_cws_status;

         end Collision_Warning_Situation_Status;
```

3. Audible_Alarm_Device

Spec

```
--
-- Audible Alarm_Device (AAD) spec
--
-- The audible_alarm device generates a tone that can be heard
-- within the host_aircraft cockpit.
--
package Audible_Alarm_Device is

   type Duration is delta 0.01 range 0.01 .. 10.00;      -- seconds
   type Frequency is range 1000 .. 10_000;               -- hertz

   procedure ring_alarm(f : in Frequency;
                        d : in Duration);

   type Alarm_Message_Type is private;

private
   type Alarm_Message_Type is
      record
         Frequency : Frequency;
         Duration : Duration;
      end record;

end Audible_Alarm_Device;
```

Documentation Components

1. ATD/CWM Software Requirements Specification (SRS)

NOTE: Only a portion of the ATD/CWM Software Requirements Specification document is shown
here to reduce the size of the ATD/CWM case study documentation. The "...." indicates
portions purposely omitted within sections of this document.

Air Traffic Control / Collision Warning Monitor Software Requirements Specification
ATD/CWM-SRS-1.0: Volume 1 of 1

1.0 : May 20, 1992

SOFTWARE REQUIREMENTS SPECIFICATION

FOR THE

ATD/CWM COMPUTER SOFTWARE CONFIGURATION ITEM

OF

THE AIR TRAFFIC CONTROL / COLLISION WARNING MONITOR SYSTEM

CONTRACT NO. Contract_1

CDRL SEQUENCE NO. XYZ004

Prepared for:

Government Agency – GA1

Prepared by:

Software Productivity Consortium
SPC Building
2214 Rock Hill Rd.
Herndon, VA 22070

Authenticated by _____   Approved by _____
(Contracting agency)   (Contractor)

Date _____   Date _____

1. Scope

This section identifies the computer software configuration item (CSCI), which briefly states the purpose of the system, describes the role of the CSCI within the system, and summarizes the purpose and content of this software requirements specification (SRS).

1.1. Identification

This SRS establishes the requirements for the CSCI identified as:

- System title: ATD/CWM

- System mnemonic: ATD/CWM

- System Identification number: ATD/CWM System_1

- CSCI title: Air–Traffic–Display / Collision–Warning–Monitor

- CSCI mnemonic: ATD/CWM

- CSCI number: XXXX

1.2. CSCI Overview

The ATD/CWM system monitors air traffic to detect collision warning situations within a surrounding surveillance area. The ATD/CWM CSCI will provide the following capabilities:

- Potential_Threat monitoring. Monitors potential threat flight characteristics ground track. relative bearing. range altitude, airspeed, and climb rate within the surveillance area.

- Intersection monitoring. Monitors the probable intersection of all aircraft with the host aircraft.

- Collision warning situation detection. Detects collision warning situations with respect to each potential threat based upon its predicted flight path and the separation minima.

- Display a corrective action advisory message on the host aircraft's display which describes what maneuvers the host aircraft should perform to avoid a collision.

- Sound an audible alarm within the host aircraft's cockpit for a detected collision warning situation.

- Transmit messages to the nearby potential threat for a detected collision warning situation.

- Transmit a message to a nearby air traffic control center for a detected collision warning situation.

1.3. Document Overview

.....

2. APPLICABLE DOCUMENTS

This section states document precedence and lists all documents referenced in this specification.

2.1. Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

MIL-STD-1815A-1983   Reference Manual For the Ada Programming Language

Copies of specifications, standards, drawings, and publications required by suppliers in connection with specified procurement functions should be obtained from the contracting agency or as directed by the contracting officer.

2.2. Non-Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

3. ENGINEERING REQUIREMENTS

This section contains the external interface and capability requirements for the ATD/CWM CSCI and identifies internal CSCI interfaces. It also contains requirements for CSCI data elements, adaptation, sizing and timing. safety. security, design constraints, software quality factors, and human performance/human engineering.

....

3.1. CSCI External Interface Requirements

The ATD/CWM CSCI will input and output data to the following external components:

- Navigation (NAV)

- Radar (RADAR)

- Audible_Alarm (AA)

- Communication (COMM)

- Air_Traffic_Display (ATD)

- Air_Traffic_Control (ATC)

APPENDIX A. A SEMI-FORMAL REQUIREMENTS METHOD

A.1 INTRODUCTION

This appendix describes the requirements method used in the third and fourth iterations of the Air Traffic Display/Collision Warning Monitor (ATD/CWM) domain case study of Synthesis practice. This requirements method was designed for precise definition of the requirements of systems that are members of the family comprising the ATD/CWM domain. This method is a variation of the method developed by the Naval Research Laboratory (NRL) Software Cost Reduction (SCR) project for defining requirements of operational flight software for the Navy's A-7 aircraft (Heninger et al. 1978). Both methods are semiformal in that the structure and form of specifications are precise but content may sometimes be informal. This method will be superseded as the Consortium requirements engineering method (Faulk et al. 1991) matures.

A.2 TERMINOLOGY

A knowledge of the following terms is required to understand the description of this requirements method.

Control function	A description of software that coordinates the activation of other functions.
Device	A physical entity with which the software interacts to acquire information or to effect required behavior.
Environment	The physical framework in which the software operates.
Function	(1) Output function (SCR); (2) Output function, control function or input function.
Input data item	A specification of an allowed transmission of data from a device to the software.
Input function	A description of software that determines the response to an input data item.
Output data item	A specification of an allowed transmission of data from the software to a device.
Output function	A description of software that is solely responsible for determining the value of one output data item.
(the) software (system)	An implementation of required behavior.

A.3 METHOD CONTRASTS

NOTE: Readers not familiar with the SCR requirements methods do not have to read this section.

The SCR method was designed to address the particular problems of complex, real-time software systems. It had six objectives (Heninger 1980) of which two are particularly relevant for the ATD/CWM domain case study:

- To specify external behavior only.

- To specify constraints on the implementation.

The method described here was designed to adhere to these objectives and all guiding principles of the SCR method but in a variant form that is appropriate to the ATD/CWM domain. The primary causes for modification are:

- A need to describe modifiable device outputs (i.e., the device, such as a CRT, maintains the state of a continuous output so that arbitrary fragments can be modified independently).

- A need to describe processing of input data items that is dependent on the output state of the software.

- A need to describe control functions whose purpose is the explicit coordination of other functions.

- A need to define the relevant theory upon which the system is built in a precise form (i.e., the precise form can be used to derive code).

In addition, some features of the SCR method are omitted from this method to simplify assumptions about the ATD/CWM problems to be described. This is possible only because this case study illustrates key facets of developing a domain and that objective is not c pendent on total realism in problem complexity. The features of the SCR method that are simplified here include:

- System states are assumed to be simple enough to describe which modes are not needed to describe output functions.

- Timing and accuracy constraints are given less attention than would otherwise be necessary. Only categories of constraint description that are essential to illustrating domain development are covered.

- Undesired event descriptions are not considered. This is an important but simple extension from SCR to this method.

- Required subsets and expected change are not treated within this method. These are accommodated under the broader domain concern for all variation: both over a single system's life cycle and among alternative systems.

A.4 STRUCTURE OF A REQUIREMENTS SPECIFICATION

The ATD/CWM requirements method guides the creation of an abstract requirements specification. An abstract specification is intended as a definitive guide for developers to use in the design.

implementation, and verification of acceptable software. It is not intended for use by end-users or others interested in an operational view of system behavior (however, such documentation should be derivable without undue effort from the abstract specification).

The requirements specification is organized into three major descriptions: theory, environment, and behavior. These descriptions are defined generally in Sections A.4.1 through A.4.3. Section A.5 provides the detailed content of each description.

A.4.1 THEORY

Theory is a description of a model of the relevant theoretical system upon which the system concept is based. This model incorporates the entities in the environment whose detectable (i.e., measurable) features and behavior can affect the behavior of the software. It further incorporates the definition of relationships among those entities and features that reveal additional information which cannot be directly measured. These entities, relationships, and features are a model of the theoretical limits within which the system must operate.

Theory is comprised of a static model and a dynamic model. The static model is a definition of the categories of information that are directly detectable in the environment. The dynamic model is a description of theoretical relationships and processes that characterize the potential behaviors of the environment, as indicated by device inputs.

A.4.2 ENVIRONMENT

Environment is a description of the external environment within which the system operates. It includes descriptions of the characteristics of the computer resources upon which the software executes and the hardware devices with which the software is required to interact. These descriptions must be sufficient to communicate all assumptions that imply constraints on the software. Each hardware device is characterized by the protocols and formats by which the software receives input data from the device and transmits output data to the device.

Environment is comprised of platform and devices. Platform defines the characteristics and assumptions of the hardware, run-time facilities, and execution primitives upon which software behavior is implemented. Devices describe the external devices (hardware, software, or hardware/software hybrids) with which the software interacts to acquire information or effect required behavior. Each device is characterized by the inputs it provides to the software and the outputs it accepts from the software.

A.4.3 BEHAVIOR

Behavior is a description of the ways in which the software can affect the environment in which it operates. This corresponds to a description of the conditions under which each possible output of data is produced and a description of how the value of the output is determined.

Behavior is composed of presentations and activities. A presentation is an output function; it describes the ways in which the software determines the value of each device output. An activity is a control function; it describes the events and conditions that enable/disable and activate/deactivate presentations and other activities for coordination among presentations and with processing of device inputs (i.e., input functions).

A.5 DETAILED CONTENT OF A REQUIREMENTS SPECIFICATION

This section contains a precise description of the content and form of required information, applicable verification criteria, and a procedure and heuristics for producing that information.

A.5.1 THEORY - STATIC MODEL

A static model is a collection of class specifications that express an essential information model of the software. A class denotes a collection of entities that are alike in specific, important ways. Differences among the entities are expressed as a set of properties that prescribe concrete data associated with each entity and links to other entities (of any class) that establish relationships among entities. A class may have subclasses that denote important subsets of the membership of the class.

Content and Form

Each class description requires the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular type of entity.

- Ancestry: a class of which this is a subclass.

- Properties: a description of the data items that characterize members of the class.

A data item description requires the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular property of members of a class.

- Value space: the data type of properties having a concrete (i.e., printable) value or designation of a class having members to which members of this class may refer.

- Description: a textual explanation of the meaning of the data item.

Verification Criteria

A completed class description must satisfy the following criteria:

- The class is uniquely named and all its properties are uniquely named relative to it and its ancestry.

- All referenced classes arc defined.

Procedure and Heuristics

To be determined

A.5.2 THEORY - DYNAMIC MODEL

A dynamic model defines logical and mathematical relationships among the concepts of the static model and derivations of additional information. These relationships and derivations form the basis

for predicting the current (or future) state of the physical system based on measurements of past (or current) state. These predictions determine the need and provide the rationale for particular software behavior.

Content and Form

A relationship or derivation description requires the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to this relationship or derivation.

- Equation: a logical or mathematical equation that defines this relationship or derivation as a function of properties in the static model (and a function of other intermediate relationships and derivations).

- Subordinate relationships/derivations: a set of relationships/derivations that are referenced only within the containing relationship/derivation.

The equation may define a time-variant or iterative computation process.

Verification Criteria

A completed relationship/derivation description must satisfy the following criteria:

- The relationship/derivation is uniquely named and subordinates are uniquely named relative to it.

- All referenced data items, relationships, and derivations are defined.

Procedure and Heuristics

To be determined

A.5.3 ENVIRONMENT – PLATFORM

Platform is a description of the hardware/software mechanism upon which a system operates. The characteristics of the platform determine the primitive mechanisms and capabilities by which the required behavior of the system can be achieved.

Content and Form

A platform description requires the following information:

- Hardware: the nature and variety of computational equipment upon which the system is required to operate.

- Operating software: the nature and variety of the run-time facilities available for the operation of the system.

- Computational software: the nature and variety of implementation facilities by which the system is realized.

Verification Criteria

A completed platform description must satisfy the following criterion:

- All platform characteristics must be consistent with the required capabilities of the system.

Procedure and Heuristics

To be determined

A.5.4 ENVIRONMENT – DEVICES

A device defines a description of the means by which a system senses or affects the environment in which it operates. Devices can be either hardware or software and must have a well-defined interface with which the system can communicate.

Content and Form

A device is described by the set of input and output data items that it provides for software interaction. Each device requires the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular device.

- Inputs: a description of the input data items that the device is able to deliver to the software.

- Outputs: a description of the output data items that the device is able to receive from the software.

An input data item description requires the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular input of a device.

- Value space: (1) A specification of the form in which input is delivered to the software. (2) A set of name/value space/description tuples describing the structure of a composite input data item.

- Timing characteristics: if time constrained, how long input data is available after occurrence (for interrupt signalled data) or how often input data is to be measured to attain sufficient accuracy (for polled data).

- Input mapping: a specification of the input function(s) that map input data values into entities (and their associated properties) of the static model.

An output data item description requires the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular output of a device.

- Value space: (1) A specification of the form in which output is to be received from the software. (2) A set of name/value space/description tuples describing the structure of a composite output data item.

- Timing characteristics: if time constrained, the rate at which output must be produced.

A value space description includes the assumed units of measurement, acceptable range and assumed accuracy or acceptable discrete values, prescribed data representation, and required instruction sequence or software interface. See (Heninger et al. 1978) for examples of value space descriptions.

Verification Criteria

A completed device description must satisfy the following criterion:

- The device is uniquely named and all its data items are uniquely named relative to it.

Procedure and Heuristics

To be determined

A.5.5 BEHAVIOR - PRESENTATION

A presentation is a specification for an output function. Presentations are described in terms of a standard set of paradigms characterized by the form in which information is organized for output. One presentation is defined for each output data item of each device. This method is tailored to a problem domain by establishing an applicable paradigm set that satisfies the particular needs of that domain.

Content and Form

All presentations require the following information. Particular paradigms may require additional information.

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular presentation.

- Paradigm: the paradigm that controls the way in which the presentation is described and how it will behave.

- Context: a class of information from the static model that indicates the type of data to be output and a predicate that can be applied to limit the instances of the class that determines an output value. Any data that is accessible, via references, to other information classes can also be considered part of the context.

- Mapping template: a paradigm-specific description of how context (and constant) data is mapped into the output data item within paradigm-determined constraints. A mapping may include an algorithmic transform of context data to satisfy the semantics or value form/units/ constraints of the output data item. Alternative mappings may be specified by identifying a set of mappings, each designated by an associated enabling condition (described by a predicate on the context).

- Input control: any input data items whose detection and handling are enabled only within the activation lifetime of the presentation and relative to its context. Input handling may be (1) independent of the particular enabling output, (2) relative only to the context of its enabling output, or (3) variable, under the control of the enabling output as determined by its particular paradigm, by correlation of the input to the structure of the output as defined by the mapping template.

- Lifetime: either invoked or activated. Invoked presentations produce an output and terminate; activated presentations produce an output whenever context data changes until deactivated.

Verification Criteria

A completed presentation description must satisfy the following criteria:

- All presentations are uniquely named.

- Context data is used consistent with its value space definition in the mapping to output.

- The mapped value of context data satisfies the value space definition of the output data item.

- The mapping template is completed correctly with respect to the referenced paradigm.

- All additional, paradigm-required information is provided.

- Any paradigm-specific constraints are satisfied.

Procedure and Heuristics

A presentation is specified through the following procedure:

1. Name and identify the context of the presentation.

   The output data item to be produced usually suggests an appropriate name and context.

2. Determine which standard mapping paradigm and lifetime best characterize the required output behavior.

3. Describe, in paradigm-specific terms, how context data (defined in the static model) is mapped into output data form (defined in devices).

4. Identify which, if any, input data items are enabled by this presentation.

5. Revise this or other referenced specifications to satisfy the verification criterion.

   When an inconsistency exists, it is necessary to determine whether the fault is in this specification or in the one(s) with which it conflicts.

A.5.6 BEHAVIOR - ACTIVITIES

An activity is a specification for a control function. Activities are described in terms of four control paradigms:

- *Sequence.* A set of activities are activated in a prescribed order.

- *Concurrence.* A set of activities are activated at the same time.

- *Selection.* One of a set of activities are activated depending on user choice or system state.

- *Subordination.* Activation of a controlled activity enables subsequent activation of other activities depending on user choice or system state.

All requirements methods for real-time systems must provide methods for representing the states and state changes that a system must track and to which it must respond. Current state information is expressed in terms of conditions and events. A condition is a truth-valued function (i.e., one that only takes on the values true or false) that characterizes the state of the system for some measurable period of time (e.g., altitude > 500 ft represents all states of the aircraft where its altitude is above 500 feet). More complex conditions can be described in the usual way by forming boolean expressions over simple conditions (e.g., altitude > 500 ft AND altitude < 2500 ft).

For real-time applications, the interest is not only in the current state but in those points in time associated with state changes. These moments where the value of a condition changes is called an event. Whereas conditions persist for measurable periods of time, events occur at single points in time. Events are a relation between the state before and after the change. For example, the event associated with altitude > 500 ft refers to any state change where the altitude was 500 ft or less and became greater than 500 ft. An event is represented by the following notation:

$$@T(condition)$$

This describes any moment at which there is a state change from a state in which the condition is not true to one in which it is. Thus, the event given above is written as @T(altitude > 500 ft). Similarly,

$$@F(condition)$$

denotes any moment the condition becomes false. Often, more information is needed about the state to describe an event than just what conditions have changed. The **when** clause describes an event in which one condition changes at a time when another holds.

$$@T(condition1) \text{ when } condition2$$
$$@F(condition3) \text{ when } condition4$$

An activity may be either invoked by another activity or activated by an internal or external event. Activation of an invoked activity may be conditional on a truth-valued expression concerning system state. Activities can be defined to terminate when all subactivities have terminated, upon occurrence of a disabling event, or when user choice dictates.

Content and Form

All activities require the following information:

- Name: an identifier that allows for explicit references, at other points in the specification, to a particular activity.

- Paradigm: the paradigm that controls the way in which the activity is described and how it will behave (one of sequence, concurrence, selection, or subordination).

- Context: a class of information from the static model that indicates the type of data to be output and a predicate that can be applied to limit the instances of the class that determine an output value. Any data that is accessible, via references, to other information classes can also be considered part of the context.

- Activation: invoked or specification of an activating event. These are represented using the @T or @F notation described above.

- Control set: the set of presentations and/or activities that are to be controlled and, for each, a specification of how its context is determined from activity context.

- Deactivation criteria: either subactivity termination, disabling event, or user choice.

Verification Criteria

A completed activity description must satisfy the following criteria:

- All activities are uniquely named.

- All referenced activities, presentations, and data items are defined.

- The activity is either event-activated or there is at least one invoking activity.

- If the activity is invoked, there is at least one activity somewhere in its ancestry that is event activated.

In addition, there must be at least one activity that is activated by system startup.

Procedure and Heuristics

An activity is specified through the following procedure:

1. Name and identify the context of the activity.

   The purpose of the activity usually suggests an appropriate name and context.

2. Determine which standard control paradigm and lifetime best characterize the required behavior.

3. Describe, in paradigm-specific terms, how context data (defined in the static model) is derived from activity context data for each element of the control set (defined in presentations or other activities).

4. Identify which, if any, input actions are enabled by this activity.

5. Revise this or other referenced specifications to satisfy the verification criterion.

   When an inconsistency exists, it is necessary to determine whether the fault is in this specification or in the one(s) with which it conflicts.

A.6 MAPPING A SPECIFICATION INTO AN ADARTS DESIGN

This requirements method is used in the ATD/CWM case study. Furthermore, the case study also follows the Ada-based Design Approach for Real-Time Systems (ADARTS) guidance on creating design structures (Software Productivity Consortium 1991b). The ADARTS guidance assumes that a Real-Time Structured Analysis (RTSA) requirements method has been used. As a result, the ADARTS guidance must be modified so that is can accommodate this requirements method.

A.6.1 PROCESS STRUCTURING

In the process structuring activity, ADARTS first requires the derivation of a set of concurrently executable sequential processes. Interpreting ADARTS guidance on process derivation in terms of this requirements method, derive an initial set of processes as follows:

- One for each possible instance of each static model class.

- One for each relationship/derivation of the dynamic model.

- One for each device.

- One for each device input mapping.

- One for each device output data item/presentation.

- One for each activity.

Each process in this initial set is minimal (i.e., atomic with respect to concurrency). Combine these processes according to ADARTS clustering criteria to create a refined process structure. ADARTS process clustering criteria is a sufficient guide to resolving the dependency relations even though this requirement method does not create data flow/control flow and finite state machine descriptions. This is a sufficient resolution for purposes of this case study, but fuller resolution is expected when the Consortium requirements method is integrated with ADARTS. Create process behavior specifications for each process consistent with ADARTS guidance.

A.6.2 CLASS STRUCTURING

The approach in the case study for class structuring is influenced by both (Parnas and Clements 1986) and ADARTS. In this approach, derive the information hiding structure directly from the structure of the requirements specification using the following heuristics. The term module is used below in lieu of the ADARTS term class to avoid confusion with the concept of static model classes.

- There are three top-level modules: environment hiding, behavior hiding, and software decision, following (Parnas, Clements, and Weiss 1985).

- Behavior hiding has two submodules: function drivers and shared functions.

- Environment hiding has two submodules: extended computer, characterized by the platform specification, and device interface, which has one submodule for each device specification.

- There is one function driver submodule for each device (i.e., related set of presentations). In addition, there is one submodule (or more, as appropriate) for the set of activities.

- Shared functions has a submodule each for the static model and the dynamic model. Each class of the static model becomes a submodule of the static model module.

Create the information hiding structure first even though this is contrary to ADARTS guidance on class structuring. This change is recommended because the information hiding structure determines the identity of all modules for which interface specifications are produced. This step combines the ADARTS guidance of deriving abstractions and organizing them into an information hiding structure. Synthesis subsumes the ADARTS creation of a generalization/specialization structure. This is accomplished by allowing for adaptation in a manner independent of the design method.

Create the module specifications and an assumptions-oriented dependency structure using the ADARTS guidance.

This page intentionally left blank.

APPENDIX B. PRESENTATION PARADIGMS

B.1 INTRODUCTION

This appendix describes the presentation paradigms used in the ATD/CWM Product Requirements. These paradigms characterize the form in which information is organized for output. Each paradigm is defined by one or more components. The paradigms presented in this appendix satisfy the particular needs of the ATD/CWM domain.

B.2 MAP PRESENTATION

The Map Presentation characterizes the form in which information is organized on a graphical display. This presentation is defined by a Context, Focus, Position_Attribute, Image, Labels, and Coordinate_System as described in the following table.

Component	Description
Context	The class of entity in the static model to display and a filter to determine a subset of the class entities which are actually displayed. The general form is: (class, filter) where filter is a boolean-valued expression expressed in terms of attributes of the class entities. Only those entities for which the filter is true are displayed. If the filter expression is a constant true, then all class entities are displayed.
Focus	A singleton class specifying the focus (i.e., central entity) in the display.
Position_Attribute	The location, in terms of direction and magnitude, of the entity position on the display. Magnitude and direction are relative to the focus. The focus is always at the center of the display.
Image	An image is composed of four attributes: shape, color, fill, and blinking. Values for an image are determined by evaluating a conditional. The general form is: (predicate, (shape, color, fill, blink)) where predicate is a boolean-valued expression. If the predicate is true, then the associated values are used. Only one predicate for a given context entity can be true at any given instant. Entities are displayed as icons having a particular geometric shape and color. The icon outline will be colored while its interior is black. The icon shape is portrayed in outline form. If fill is specified, then the icon interior is also colored. Icons are positioned relative to the focus. The display orientation for the triangle and square will always appear as shown in Figure B-1. The blinking rate is fixed at 0.125 seconds.

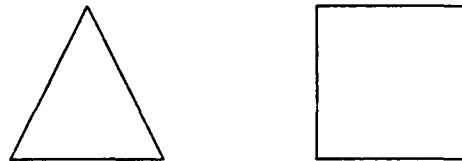| Component	Description
Labels	Textual attributes shown with the displayed object. These are displayed in a vertical field as indicated by: text_1 text_2 text_3 This field is located either immediately to the left, to the right, above, or below the displayed entity.
Coordinate_System	Maximum (x, y) range of display in nautical miles.

Figure B-1. Icon Display Orientation

Figure B-2 represents the type of display the map paradigm provides. The boxes indicate displayed entities. The direction is given by θ and the magnitude is the distance from the box to the focus. The cross-hatch (+) marks the location of the focus.
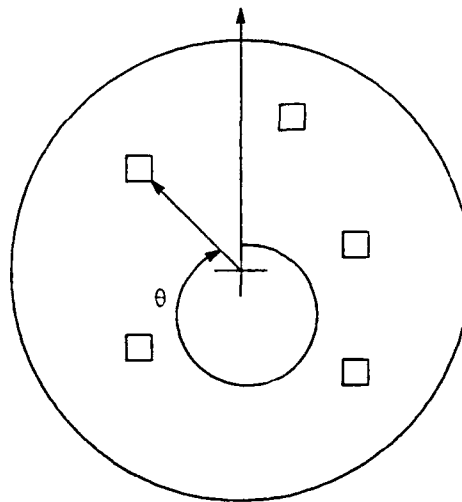
Figure B-2. Map Presentation

B.3 TEXT PRESENTATION

The Text Presentation characterizes the form in which printable textual information is organized on a display. This presentation is defined by a Template and Context as described in the following table.

Component	Description
Template	Boilerplate text having embedded placeholders that must be filled in before the message is sent to the device. A value for the boilerplate text is determined by evaluating a conditional of the form: (predicate, value) where predicate is a boolean-valued expression. If the predicate is true, then the associated value is used for the boilerplate. There is a list of these (predicate, value) pairs for template. Only one predicate for a given context entity can be true. Text car be either alphanumeric or hexadecimal constants of the form 0xNN (N is a hexadecimal digit 0–9, A–F). Placeholders are designated by %attribute where attribute indicates the source of information within the context entity. Placeholders are filled in using the corresponding attribute value.
Context	The class of entity in the static model from which to retrieve the information to fill in the placeholders.

When activated, this presentation formats the text from the template replacing placeholders with information retrieved from the attributes of the context class.

B.4 AUDIBLE ALARM PRESENTATION

The Audible Alarm Presentation characterizes the pitch and duration for an audible alarm. This presentation is defined by a Context and Freq_and_Duration as described in the following table.

Component	Description
Context	The class of entity in the static model from which to determine the frequency of the audible alarm and how long (i.e., duration) to ring it.
Freq_and_Duration	The pitch of the alarm measured in hertz (resolution of one hertz) and how long the alarm is rung measured in seconds (resolution of 0.01 seconds). A value for frequency and duration is determined by evaluating a conditional of the form: (predicate, frequency, duration) where predicate is a boolean-valued expression. If the predicate is true, then the associated frequency and duration is used. There is a list of these (predicate, frequency, duration) triples for frequency and duration. Only one predicate for a given context entity can be true.

When activated, this presentation causes the audible alarm to ring at a specified frequency for a specified time duration by accessing the context entity to determine which frequency and duration predicates are true.

B.5 BINARY PRESENTATION

The Binary Presentation characterizes the form in which information is organized into a binary representation. This presentation is defined by a Context and Template as described in the following table.

Component	Description
Context	The class of entity in the static model from which to retrieve the information to fill in the placeholders.
Template	Boilerplate having embedded placeholders that must be filled in before the message is sent to the device. Placeholders are designated by %attribute where attribute indicates the source of information within the context entity. Placeholders are also designated by @class.attribute signifying where to get information from a singleton class. The @ can only be used for a singleton class. All numeric information in the template, including the values filled in for the placeholders, is converted to binary. A value for the boilerplate is determined by evaluating a conditional of the form: (predicate, value) where predicate is a boolean-valued expression. If the predicate is true, then the associated value is used for the boilerplate. There is a list of (predicate, value) pairs for template. Only one predicate for a given context entity can be true.

When activated, this presentation formats the information from the template replacing placeholders with information retrieved from the attributes of the context class.

APPENDIX C. AIR TRAFFIC DISPLAY/COLLISION WARNING MONITOR CASE STUDY WITH AUTOMATION

C.1 INTRODUCTION

This section describes how a commercially available tool—MetaTool™ Specification-Driven-Tool Builder (SDTB [AT&T 1990])—could be used to automate a portion of the Synthesis Domain Implementation and Application Engineering activities for a domain. Even though effective use of Synthesis does not depend on automation. the mechanical orientation of Synthesis is partially predicated on automation of the Application Engineering process. As context for the remainder of the section, the following two subsections briefly describe MetaTool SDTB and the general strategy of the Consortium for using it for the ATD/CWM domain. Sections C.2 through C.6 present sample work products for that domain, including examples of the MetaTool source and product description files and examples of products built using the ATD/CWM Specification-Driven tool (ATD/CWM SDTool) produced with MetaTool SDTB. The notation used in the MetaTool source and product description files is defined in (AT&T 1990); however, the examples given can be understood without detailed knowledge of this notation.

C.1.1 MetaTool Specification-Driven-Tool Builder Overview

MetaTool SDTB translates tool description files into specification-driven tools (SDTools). Similarly, generated SDTools translate textual specifications into text-based products such as code and documentation. MetaTool SDTB reads a source description file and a product description file in order to automatically generate an SDTool (Figure C-1). Similarly, a generated SDTool reads a specification file and produces products according to the information contained in that specification.
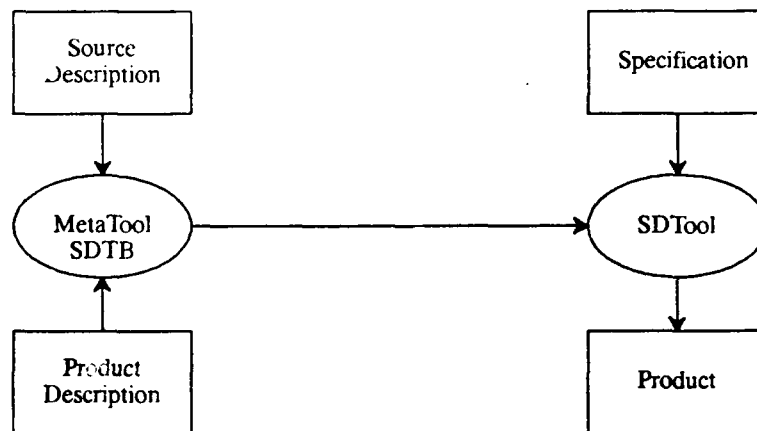

Figure C-1. SDTool Development Using MetaTool Specification-Driven-Tool Builder

A source description file describes the grammar for a language to be used in the corresponding SDTool specification file. A product description file defines a template that describes how the SDTool is to build a product depending on the information the SDTool finds in its input specification file.

Each SDTool built by MetaTool SDTB is composed of three functional parts: the front end, the middle part, and the back end (Figure C-2 from [AT&T 1990]). The front end parses the input specification and creates an internal parse tree. MetaTool SDTB automatically generates the front end from the source description file. The middle part executes after the front end and performs whatever operations are necessary to prepare for generation of the products. The middle part is a customized part provided by the SDTool developer. The back end of a SDTool generates the products. MetaTool SDTB automatically generates the back end from the product description files.
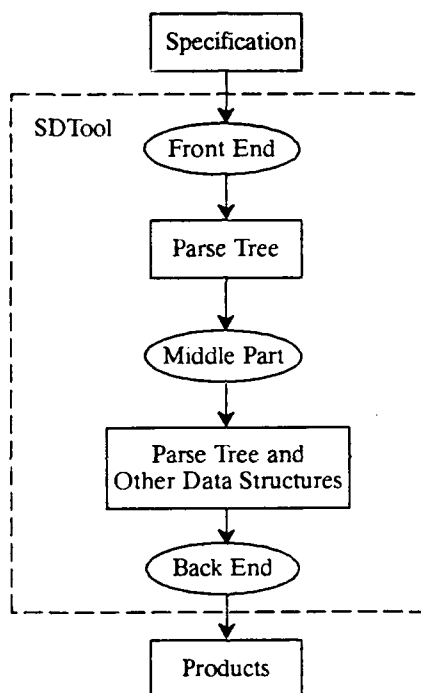
Figure C-2. Internal Structure of an SDTool

C.1.2 Using MetaTool Specification-Driven-Tool Builder to Support the Air Traffic Display/Collision Warning Monitor Domain

The Product Implementation activity defines a product called Generation Procedure. This product defines a mechanical process for selecting, adapting, and composing adaptable components based on decisions expressed in an Application Model to form an application. The Generation Procedures also define a mapping of decisions in the Application Model to parameters of adaptable components. The Product Implementation activity also creates adaptable components that satisfy the product design specifications. The Consortium used MetaTool SDTB to automate only the mechanical selection and adaptation process, and the decision mapping aspects of the ATD/CWM Generation Procedures. In addition, MetaTool SDTB provides a means to represent adaptable components.

To realize the mechanical process, the ATD/CWM SDTool accepts an ATD/CWM Application Model expressed in textual form and uses information contained in the model to determine which code and

documentation components to generate. The grammar of the ATD/CWM Application Model Notation is expressed in a Backus-Naur form (BNF) in a MetaTool source description file. The source description file also defines all Adaptable Components that the ATD/CWM SDTool could select and adapt. Implementations of the Adaptable Components are captured in product description files (Figure C-3). The product description files contain metaprogramming constructs used for both Generation Implementation (i.e., defining a Generation Procedure for selecting and adapting a component) and Component Implementation (i.e., defining an Adaptable Component).
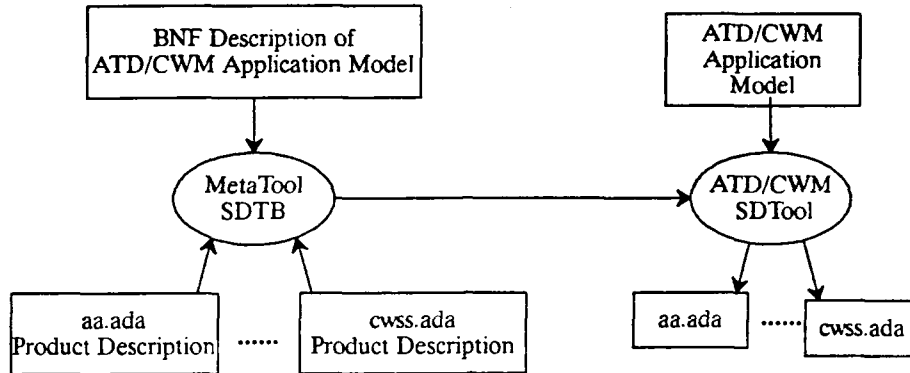
Figure C-3. Air Traffic Display/Collision Warning Monitor SDTool Development Using MetaTool Specification-Driven-Tool Builder

To implement the decision mapping. the Consortium developed customized code for the middle part of the ATD/CWM SDTool. This code calculates values for parameters of the Adaptable Components by extracting decisions of the ATD/CWM Application Model from the internal parse tree representation of the Application Model. Figure C-4 depicts the resulting ATD/CWM SDTool.
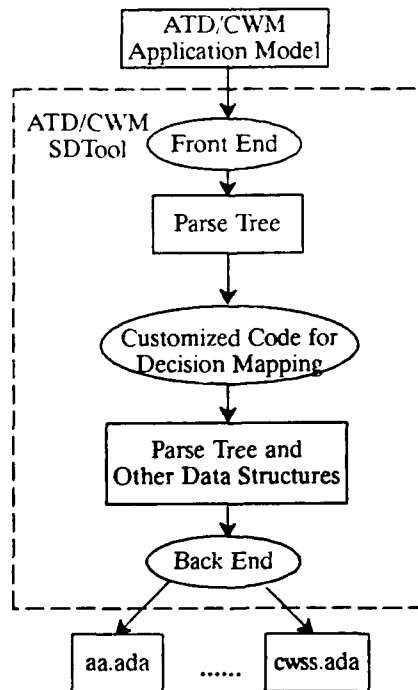

Figure C-4. Internal Structure of Air Traffic Display/Collision Warning Monitor SDTool

The ATD/CWM SDTool automates a portion of the ATD/CWM Application Engineering Process Support—namely, a portion of the Application Production phase (partially shaded box in Figure C-5). Application Modeling, which precedes Application Production, remains a manual process. The application engineer must follow a mechanical process to build and assess an Application Model. Once the application engineer is satisfied that the Application Model describes the desired ATD/CWM system, he uses the ATD/CWM SDTool to produce the desired components for that system. The application engineer then manually composes these components to obtain the desired products (e.g., application, documentation). The Consortium could have used MetaTool SDTB to compose the components as well.
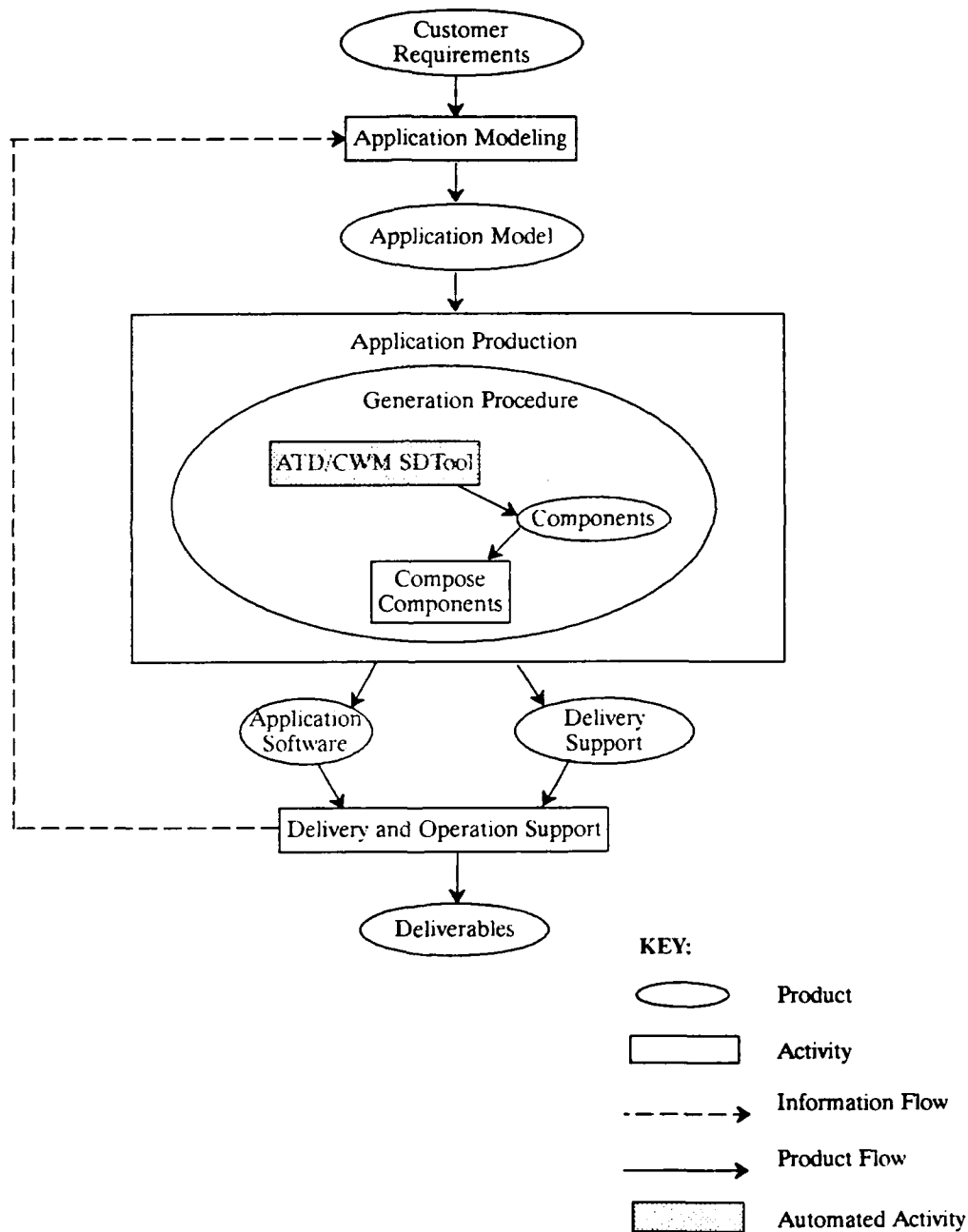

Figure C-5. Partial Automation of Air Traffic Display/Collision Warning Monitor Application Engineering Process

C.2 GENERATION PROCEDURES AUTOMATION

The source description file describes the BNF grammar of the Application Modeling Notation that the ATD/CWM SDTool accepts. This is the language used to write an ATD/CWM Application Model. The source description file also identifies the products that the ATD/CWM SDTool generates. An excerpt of the source description file is shown in Figure C-6.

```
%grammar

model :    (model_name project_information host_aircraft potential_threat cws+)

model_name : ("Application_Model_Name:" id)

project_information : ("Project_Information:" contract system_info)

. . .

host_aircraft : ("Host_Aircraft_Characteristics:" surveillance_area
                                                   host_aircraft_shape)

surveillance_area :    ("Surveillance_Area:" number)

host_aircraft_shape : ("Host_Aircraft_Shape:" icon_shape)

. . .



%product aa.ada
%product cwss.ada
. . .
```

Figure C-6. Generation Procedures Source Specification (Excerpt)

Each %product statement (e.g., %product aa.ada) identifies a product description file which defines a product of the ATD/CWM SDTool. Each product description file corresponds to an Adaptable Component specified in the Product Design Activity. Figure C-6 shows a partial listing of the products produced by the ATD/CWM SDTool.

The product description file contains a template that describes how to build the product. The template contains target text and metaprogramming constructs. Target text (any text not escaped with the % MetaTool SDTB meta-character) appears exactly as is in the generated product. The metaprogramming constructs specify how to adapt the component. Metaprogramming constructs begin with the % character. Figure C-7 shows an excerpt of the product description file for product aa.ada. You can compare this implementation of aa.ada with the TRF2 implementation in the fourth iteration of the ATD/CWM case study.

```
%template
%if _ring != NULL %then
--
-- Audible_Alarm (AA) body
--
-- This module determines the characteristics at which
-- to ring the audible alarm for a specified collision warning
-- situation.
--
with Potential_Threat;
with Audible_Alarm_Device;
package body Audible_Alarm is

    procedure ring_alarm(cws : in Potential_Threat.cws_id)
    is
    begin
        case cws is
%for (r=_ring; r != NULL; r = r->_next) %loop
            when Potential_Threat.%s(r->_cws_name) =>
                Audible_Alarm_Device.ring_alarm(f => %s(r->_frequency),
                                                d => %s(r->_duration));
%end-loop
            when others =>
                return;
        end case;
    end ring_alarm;

end Audible_Alarm;
%end-if
```

Figure C-7. Product Description File — aa.ada (Excerpt)

C.3 GENERATED PRODUCTS

This section shows samples of the aa.ada and cwss.ada products produced by the ATD/CWM SDTool. These samples are based on the ATD/CWM Application Model excerpt shown in Figure C-8. The front end of the ATD/CWM SDTool parses this textual input producing in internal parse tree. The middle code calculates the parameters for the adaptable components using information contained in the Application Model. The back end generates the desired work products.

The ATD/CWM SDTool will select and generate product aa.ada because the Application Model contains a Situation_Response which has an Alarm of Yes (i.e., the ATD/CWM SDTool evaluates the %if conditional at the beginning of the %template and determines that it is true). The ATD/CWM SDTool will also adapt this component based on values of decisions expressed in the Application Model (e.g., alarm pitch and duration). An excerpt of the adapted aa.ada component is shown in Figure C-9. The ATD/CWM SDTool will always select cwss.ada. Furthermore, this component is also adapted based on values in the Application Model (e.g., Min and Max time values) as shown in Figure C-10.

```
Collision_Warning_Situation:
   Collision_Warning_Situation_Name:      Possible

   Situation_Definition:
      Situation_Flight_Characteristics:
         Time:
            Min:        60
            Max:        90
         Range:
            Range_Min:
            Range_Max:
      Situation_Aircraft_Partition:   ALL
      Situation_Severity:             0.20

   Situation_Response:
      Response_to_ATC:                No
      Response_to_other_Aircraft:     No
      Corrective_Action_Response:     No
      Alarm:                          Yes

      Alarm_Characteristics:
         Pitch:                  2500
         Duration:               5.00
```

Figure C-8. Air Traffic Display/Collision Warning Monitor Application Model (Excerpt)

```
procedure ring_alarm(cws : in Potential_Threat.cws_id)
is
begin
   case cws is
      when Potential_Threat.Possible =>
         Audible_Alarm_Device.ring_alarm(f => 2500, d => 5.00);
      when ...
         ...
      when ...
         ...
      when others =>
         return;
   end case;
end ring_alarm;
```

Figure C-9. Generated Product — aa.ada (Excerpt)

```
    function get_cws_status(threat : in Potential_Threat.pt_handle)
                                    return Potential_Threat.cws_id
    is
        time_to_intersect : Physical_Quantities.seconds;
        target_range : Physical_Quantities.nautical_mile;
    begin
        target_range := Potential_Threat.get_range(threat);
        time_to_intersect := Situation_Dynamics.get_elapsed_time(threat);
        if (...) then
            ...
        elsif (...) then
            ...
        elsif (
            (60.0 <= time_to_intersect and then
                        time_to_intersect < 90.0)) then
                return Potential_Threat.Possible;
        elsif (...) then
            ...
        else
            return Potential_Threat.normal;
        end if;
    end get_cws_status;
```

Figure C-10. Generated Product — cwss.ada (Excerpt)

C.4 METATOOL SPECIFICATION-DRIVEN-TOOL BUILDER DESCRIPTION FILES

This section provides complete listings of the source description file (defining the grammar for the ATD/CWM Application Model) and product descriptions files for aa.ada and cwss.ada.

C.4.1 SOURCE DESCRIPTION FILE

```
%features R

%red id number

%lex-definitions
%%e 1500
%%p 5000
%%n 1000
%%a 4000

%grammar

model :   (model_name project_information host_aircraft potential_threat cws+)

model_name : ("Application_Model_Name:" id)

project_information : ("Project_Information:" contract system_info)

contract :   ("Contract:" contract_agency contract_number contract_cdrl)

contract_agency :   ("Agency:" id)

contract_number :   ("Number:" id)

contract_cdrl : ("CDRL:" id)

system_info :   ("System:" system_name system_mnemonic system_id)

system_name :   ("Name:" id)

system_mnemonic :   ("Mnemonic:" id)

system_id :   ("Id:" id)

host_aircraft : ("Host_Aircraft_Characteristics:" surveillance_area
                                                   host_aircraft_shape)

surveillance_area :   ("Surveillance_Area:" number)

host_aircraft_shape : ("Host_Aircraft_Shape:" icon_shape)

potential_threat : ("Potential_Threat_Characteristics:"
                                                   idreq idshape uidshape)

idreq :   ("Identification_Requirements:" id_requirement)

id_requirement :   ("(" id1:id [ "," id2:id] ")")
```

```
idshape : ("Shape_of_Identified_Aircraft:" icon_shape)

uidshape :   ("Shape_of_Unidentified_Aircraft:" icon_shape)

cws : ("Collision_Warning_Situation:" cws_name defn response display)

cws_name :   ("Collision_Warning_Situation_Name:" id)

defn : ("Situation_Definition:" predicate partition severity)

predicate :   ("Situation_Flight_Characteristics:" cws_time cws_range)

cws_time :   ("Time:" min_time max_time)

min_time :   ("Min:" [number])

max_time :   ("Max:" [number])

cws_range :   ("Range:" min_range max_range)

min_range :   ("Range_Min:" [number])

max_range :   ("Range_Max:" [number])

partition :   ("Situation_Aircraft_Partition:" id)

severity :   ("Situation_Severity:" number)

response :   ("Situation_Response:" atc pt corrective alarm)

atc : ("Response_to_ATC:" yes_or_no)

pt :   ("Response_to_other_Aircraft:" yes_or_no)

corrective : ("Corrective_Action_Response:" yes_or_no)

alarm :   ("Alarm:" id alarm_info)

alarm_info : ("Alarm_Characteristics:" alarm_pitch alarm_duration)

alarm_pitch :   ("Pitch:" [number])

alarm_duration :   ("Duration:" [number])

display : ("Situation_Display:" host_color id_color id_blink id_fill
                                        uid_color uid_blink uid_fill)

host_color : ("Color_of_Host_Aircraft:" icon_color)

id_color :   ("Color_of_Identified_Potential_Threats:" icon_color)

id_blink :   ("Blinking_Identified_Potential_Threats:" yes_or_no)

id_fill : ("Fill_Identified_Potential_Threats:" yes_or_no)

uid_color : ("Color_of_Unidentified_Potential_Threats:" icon_color)
```

```
uid_blink : ("Blinking_Unidentified_Potential_Threats:" yes_or_no)

uid_fill : ("Fill_Unidentified_Potential_Threats:" yes_or_no)

icon_shape : ("square" | "circle" | "triangle")

icon_color : ("red"| "orange"| "green"| "yellow"| "white"| "blue"| "black"|
                    "pink" | "purple" | "indigo" | "violet")

yes_or_no : ("Yes" | "No")

id :   <[a-zA-Z][_a-zA-Z0-9/]*>

number : <[0-9]+("."[0-9]*)?>

%middlecode
extract();

%files
extract.c extract.h

%makefile
# dependencies
extract.o : extract.c extract.h
GEN_1.o :    extract.h
GEN_2.o :    extract.h
GEN_4.o :    extract.h
GEN_5.o :    extract.h
GEN_a.o :    extract.h

# Use GNU C compiler (the SUN version has a bug in it)
CC= gcc

%product srs.doc    -a
%product aa_.ada    -1
%product aa.ada     -2
%product cwss_.ada -3
%product cwss.ada  -4
%product aad_.ada  -5
%product na_.ada   -7
%product na.ada    -8
```

C.4.2 PRODUCT DESCRIPTION FILE – aa.ada

```
%declare
#include "extract.h"

extern RING *_ring;

static RING *r;

%template
%if _ring != NULL %then
--
```

```
-- Audible_Alarm (AA) body
--
-- This module determines the characteristics at which
-- to ring the audible alarm for a specified collision warning
-- situation.
--
with Potential_Threat;
with Audible_Alarm_Device;
package body Audible_Alarm is

    procedure ring_alarm(cws : in Potential_Threat.cws_id)
    is
    begin
        case cws is
%for (r=_ring; r != NULL; r = r->_next) %loop
            when Potential_Threat.%s(r->_cws_name) =>
                Audible_Alarm_Device.ring_alarm(f => %s(r->_frequency),
                                                d => %s(r->_duration));
%end-loop
            when others =>
                return;
        end case;
    end ring_alarm;

end Audible_Alarm;
%end-if
```

C.4.3 PRODUCT DESCRIPTION FILE – cwss.ada

```
%declare
#include "extract.h"

extern CWS_TYPE *_cws;
extern char *_partition;

static CWS_TYPE *cws_local;
static int time_form = 0;
static int range_form = 0;
static int id_or_uid_partition = 0;

%template
--
-- Collision Warning Situation Status (CWSS)  package body
--
-- This module determines the collision warning situation status
-- for the given potential_threat and host_aircraft.
--
with Potential_Threat;
with Physical_Quantities; use Physical_Quantities;
%for (cws_local=_cws; cws_local != NULL; cws_local = cws_local->_next) %loop
    %if cws_local->_predicate.cws_def_type == TIME_ONLY %then
        %{ time_form = 1; }
    %elif cws_local->_predicate.cws_def_type == RANGE_ONLY %then
```

```
         %{ range_form = 1; }
      %else
         %{ time_form = 1;
            range_form = 1; }
      %end-if
      %if strcmp(cws_local->_partition, "ALL") != 0 %then
         %{ id_or_uid_partition = 1; }
      %end-if
%end-loop
%if time_form %then
with Situation_Dynamics;
%end-if

package body Collision_Warning_Situation_Status is

   function get_cws_status(threat : in Potential_Threat.pt_handle)
                                      return Potential_Threat.cws_id
   is
%if id_or_uid_partition %then
      partition : Potential_Threat.partition;
%end-if
%if time_form %then
      time_to_intersect : Physical_Quantities.seconds;
%end-if
%if range_form %then
      target_range : Physical_Quantities.nautical_mile;
%end-if
   begin
%if id_or_uid_partition %then
      partition := %s(_partition).get_partition(threat);
%end-if
%if range_form %then
      target_range := Potential_Threat.get_range(threat);
%end-if
%if time_form %then
      time_to_intersect := Situation_Dynamics.get_elapsed_time(threat);
%end-if
      if (
%for (cws_local=_cws; cws_local != NULL; cws_local = cws_local->_next) %loop
   %if strcmp(cws_local->_partition, "ALL") != 0 %then
         partition = Potential_Threat.%s(cws_local->_partition) and then
   %end-if
   %if cws_local->_predicate.cws_def_type == RANGE_ONLY %then
         (%s(cws_local->_predicate.range_min) <= target_range and then
target_range < %s(cws_local->_predicate.range_max))) then
            return Potential_Threat.%s(cws_local->_cws_name);
   %elif cws_local->_predicate.cws_def_type == TIME_ONLY %then
         (%s(cws_local->_predicate.time_min) <= time_to_intersect and then
                     time_to_intersect < %s(cws_local->_predicate.time_max)))
then
            return Potential_Threat.%s(cws_local->_cws_name);
   %else
         ((%s(cws_local->_predicate.range_min) <= target_range and then
```

```
target_range <
                                            %s(cws_local->_predicate.range_max))
         or else
         (%s(cws_local->_predicate.time_min) <= time_to_intersect and then
                      time_to_intersect <
%s(cws_local->_predicate.time_max)))) then
         return Potential_Threat.%s(cws_local->_cws_name);
   %end-if
   %if cws_local->_next != NULL %then
      elsif (
   %end-if
%end-loop
      else
         return Potential_Threat.normal;
      end if;
   end get_cws_status;

   --
   -- Determine the collision warning situation status of
   -- the host_aircraft.
   --
   function get_host_status return Potential_Threat.cws_id
   is
   begin
   --     ...
   end get_host_status;

end Collision_Warning_Situation_Status;
```

C.5 AIR TRAFFIC DISPLAY/COLLISION WARNING MONITOR APPLICATION MODEL

This section shows a complete ATD/CWM Application Model that the ATD/CWM SDTool will accept.

```
Application_Model_Name:   ATD/CWM_System

Project_Information:
   Contract:
      Agency:             SPC
      Number:             A123_456_789
      CDRL:               AA_BB_CC
   System:
      Name:             ATD_CWM_N
      Mnemonic:         ATD_CWM_M
      Id:               ATD_CWM_I

Host_Aircraft_Characteristics:
   Surveillance_Area:            125
   Host_Aircraft_Shape:          circle

Potential_Threat_Characteristics:
   Identification_Requirements:        (airspeed)
   Shape_of_Identified_Aircraft:       triangle
   Shape_of_Unidentified_Aircraft:     square
```

```
Collision_Warning_Situation:
   Collision_Warning_Situation_Name:     Monitored

   Situation_Definition:
      Situation_Flight_Characteristics:
         Time:
            Min:
            Max:
         Range:
            Range_Min:              0.0
            Range_Max:              125.0
      Situation_Aircraft_Partition:      ALL
      Situation_Severity:          0.10

   Situation_Response:
      Response_to_ATC:                No
      Response_to_other_Aircraft:     No
      Corrective_Action_Response:     No
      Alarm:                          No

      Alarm_Characteristics:
         Pitch:
         Duration:

   Situation_Display:
      Color_of_Host_Aircraft:                      white
      Color_of_Identified_Potential_Threats:    orange
      Blinking_Identified_Potential_Threats:    No
      Fill_Identified_Potential_Threats:        No
      Color_of_Unidentified_Potential_Threats: green
      Blinking_Unidentified_Potential_Threats: No
      Fill_Unidentified_Potential_Threats:      No

Collision_Warning_Situation:
   Collision_Warning_Situation_Name:     Possible

   Situation_Definition:
      Situation_Flight_Characteristics:
         Time:
            Min:              60.0
            Max:              90.0
         Range:
            Range_Min:
            Range_Max:
      Situation_Aircraft_Partition:      ALL
      Situation_Severity:          0.20

   Situation_Response:
      Response_to_ATC:                Yes
      Response_to_other_Aircraft:     No
      Corrective_Action_Response:     No
      Alarm:                          Yes

      Alarm_Characteristics:
         Pitch:             2500
         Duration:          5.00
```

```
    Situation_Display:
        Color_of_Host_Aircraft:                     white
        Color_of_Identified_Potential_Threats:      orange
        Blinking_Identified_Potential_Threats:      No
        Fill_Identified_Potential_Threats:          No
        Color_of_Unidentified_Potential_Threats:    green
        Blinking_Unidentified_Potential_Threats:    No
        Fill_Unidentified_Potential_Threats:        No

 Collision_Warning_Situation:
    Collision_Warning_Situation_Name:       Potential

    Situation_Definition:
        Situation_Flight_Characteristics:
            Time:
                Min:            30.0
                Max:            60.0
            Range:
                Range_Min:
                Range_Max:
            Situation_Aircraft_Partition:       ALL
            Situation_Severity:             0.30

    Situation_Response:
        Response_to_ATC:                    Yes
        Response_to_other_Aircraft:         Yes
        Corrective_Action_Response:         Yes
        Alarm:                  Yes

        Alarm_Characteristics:
            Pitch:              4000
            Duration:           5.00

    Situation_Display:
        Color_of_Host_Aircraft:                     white
        Color_of_Identified_Potential_Threats:      pink
        Blinking_Identified_Potential_Threats:      Yes
        Fill_Identified_Potential_Threats:          Yes
        Color_of_Unidentified_Potential_Threats:    blue
        Blinking_Unidentified_Potential_Threats:    Yes
        Fill_Unidentified_Potential_Threats:        Yes

 Collision_Warning_Situation:
    Collision_Warning_Situation_Name:       Imminent

    Situation_Definition:
        Situation_Flight_Characteristics:
            Time:
                Min:            0.0
                Max:            30.0
            Range:
                Range_Min:
                Range_Max:
            Situation_Aircraft_Partition:       ALL
            Situation_Severity:             0.50
```

```
Situation_Response:
   Response_to_ATC:                     Yes
   Response_to_other_Aircraft:          Yes
   Corrective_Action_Response:          Yes
   Alarm:               Yes

   Alarm_Characteristics:
      Pitch:             8000
      Duration:          5.00

Situation_Display:
   Color_of_Host_Aircraft:                   red
   Color_of_Identified_Potential_Threats:    yellow
   Blinking_Identified_Potential_Threats:    Yes
   Fill_Identified_Potential_Threats:        Yes
   Color_of_Unidentified_Potential_Threats:  purple
   Blinking_Unidentified_Potential_Threats:  Yes
   Fill_Unidentified_Potential_Threats:      Yes
```

C.6 GENERATED PRODUCTS

This section shows the complete products aa.ada and cwss.ada produced by the ATD/CWM SDTool using the Application Model shown in Section C.5.

C.6.1 PRODUCT – aa.ada

```
--
-- Audible_Alarm (AA) body
--
-- This module determines the characteristics at which
-- to ring the audible alarm for a specified collision warning
-- situation.
--
with Potential_Threat;
with Audible_Alarm_Device;
package body Audible_Alarm is

   procedure ring_alarm(cws : in Potential_Threat.cws_id)
   is
   begin
      case cws is
         when Potential_Threat.Imminent =>
            Audible_Alarm_Device.ring_alarm(f => 8000, d => 5.00);
         when Potential_Threat.Potential =>
            Audible_Alarm_Device.ring_alarm(f => 4000, d => 5.00);
         when Potential_Threat.Possible =>
            Audible_Alarm_Device.ring_alarm(f => 2500, d => 5.00);
         when others =>
            return;
      end case;
   end ring_alarm;

end Audible_Alarm;
```

C.6.2 Product – cwss.ada

```
--
-- Collision Warning Situation Status (CWSS)  package body
--
-- This module determines the collision warning situation status
-- for the given potential_threat and host_aircraft.
--
with Potential_Threat;
with Physical_Quantities; use Physical_Quantities;
with Situation_Dynamics;

package body Collision_Warning_Situation_Status is

    function get_cws_status(threat : in Potential_Threat.pt_handle)
                                        return Potential_Threat.cws_id
    is
       time_to_intersect : Physical_Quantities.seconds;
       target_range : Physical_Quantities.nautical_mile;
    begin
       target_range := Potential_Threat.get_range(threat);
       time_to_intersect := Situation_Dynamics.get_elapsed_time(threat);
       if (
          (0.0 <= time_to_intersect and then
                     time_to_intersect < 30.0)) then
              return Potential_Threat.Imminent;
       elsif (
          (30.0 <= time_to_intersect and then
                     time_to_intersect < 60.0)) then
              return Potential_Threat.Potential;
       elsif (
          (60.0 <= time_to_intersect and then
                     time_to_intersect < 90.0)) then
              return Potential_Threat.Possible;
       elsif (
          (0.0 <= target_range and then target_range < 125.0)) then
              return Potential_Threat.Monitored;
       else
          return Potential_Threat.normal;
       end if;
    end get_cws_status;


--
-- Determine the collision warning situation status of
-- the host_aircraft.
--
    function get_host_status return Potential_Threat.cws_id
    is
    begin
--     ...
    end get_host_status;

end Collision_Warning_Situation_Status;
```

APPENDIX D. AIR TRAFFIC DISPLAY/COLLISION WARNING MONITOR CUSTOMER REQUIREMENTS

1. Introduction

Create an Air Traffic Display/Collision Warning Monitor. The display should be continuously updated (at least four (4) times per second) to show all aircraft within an 125 mile radius. In addition, the monitor will immediately respond to warning situations with appropriate alarms and actions.

1.1 Air Traffic Display (ATD) System

The ATD will display items in three (3) uniquely determined classes:

1. Host Aircraft – This display item will show the following information:

   a. Altitude (in feet)

   b. Air Speed (in knots)

   c. Course Bearing (degrees)

   d. Aircraft ID (alphanumeric)

2. Other identifiable aircraft – These display items will show the following information for each aircraft:

   a. Altitude (in feet)

   b. Air Speed (in knots)

   c. Course Bearing (degrees)

   d. Aircraft ID (alphanumeric)

This information shall be obtained from the following sources:

   i. Target aircraft

   ii. Nearby air traffic control center

3. Unidentifiable aircraft – These display items will show as much of the information required in (2.) as possible. (unidentifiable is defined to mean that the airspeed of (2.) is missing.)

1.2 Collision Warning Monitor

The Collision Warning Monitor (CWM) System will evaluate the host aircraft's course in relation to all other air traffic within an 125 mile radius. The CWM will recognize three (3) types of warning situations and will take the action indicated below for each.

1. Possible Collision – A collision between the host aircraft and one or more target aircraft is considered possible if the two aircraft could possibly intersect in less than 90 seconds. For aircraft which are identified the time to collision would be based on the time to impact taking into consideration any course change to an intersection course , with the target aircraft maintaining the current speed. For unidentified aircraft, the time to collision would be based on the time to impact assuming a course which intersected that of the host aircraft at a worst case target aircraft speed. In the event of a possible collision. the CWM will take the following actions:

   a. The ATD system icon representing the aircraft will be tagged as specified in Paragraph 2.1.

   b. An audible alarm will be sounded in the cockpit.

   c. A message will be sent to the nearest air traffic control center.

2. Potential Collision – A collision between the host aircraft and one or more target aircraft is considered potential if a collision could possibly intersect in less than 60 seconds. For aircraft which are identified the time to collision would be based on the time to impact taking into consideration any course change to an intersection course , with the target aircraft maintaining the current speed. For unidentified aircraft, the time to collision would be based on the time to impact assuming a course which intersected that of the host aircraft at a worst case target aircraft speed. In the event of a potential collision. the CWM will take the following actions:

   a. The ATD system icon representing the aircraft will change color and begin to blink at twice the update rate.

   b. An audible alarm will be sounded in the cockpit.

   c. A message will be sent to the nearest air traffic control center.

   d. A message will sent to the other aircraft.

   e. A corrective action advisory message will be displayed on the ATD.

3. Imminent Collision – A collision between the host aircraft and one or more target aircraft is considered imminent if a collision could occur in less than 30 seconds. For aircraft which are identified the time to collision would be based on the time to impact taking into consideration any course change to an intersection course, with the target aircraft maintaining the current speed. For unidentified aircraft, the time to collision would be based on the time to

impact assuming a course which intersected that of the host aircraft at a worst case target aircraft speed. In the event of an potential collision, the CWM will take the following actions:

    a. The ATD system icon representing the aircraft will change color and begin to blink at twice the update rate.

    b. An audible alarm will be sounded in the cockpit.

    c. A message will be sent to the nearest air traffic control center.

    d. A message will sent to the other aircraft.

    e. A corrective action advisory message will be displayed on the ATD.

    f. The ATD system icon for the host aircraft will change color.

Note that these warning situations have been listed in reverse priority order.

2. ATD Display Item Data

  2.1 Identifying Icons

    1. The host aircraft will be identified by a circle.

    2. Identifiable aircraft will be shown as triangles.

    3. Unidentifiable aircraft will be shown as squares.

    4. Aircraft involved in a potential collision warning will be tagged by filling in their respective icon. This state will be maintained until the target aircraft moves to a condition of lower priority than that of a potential collision.

  2.2 Colors

    1. The host aircraft will be displayed in white under most conditions. It will be displayed in red if an imminent collision situation occurs.

    2. Identifiable aircraft will be displayed in orange under normal conditions. They will be displayed in pink if they are involved in a potential collision situation, yellow if an imminent collision situation occurs.

    3. Unidentifiable aircraft will be displayed in green under normal conditions. They will be displayed in blue if they are involved in a potential collision situation, and purple if an imminent collision situation occurs.

  2.3 Position of aircraft icons (other than host)

    1. Icons will be positions relative to the host aircraft position on the screen based on least cluttered location for unidentified aircraft.

    2. After initial positioning of the aircraft, it will continue to move along the radius of the concentric rings until it goes out of range.

3. Alarms

A distinct audible alarm will be issued for each of the warning situations described in Paragraph 1.2 whenever a target aircraft goes from a state of lower priority to one of higher priority.

4. Assumptions

The ATD/CWM System currently does not have the capability to handle multiple warnings for the host aircraft. Therefore, the system will always react to warning situations in priority order.

REFERENCES

Aircraft Owners and
Pilots Association
1990

AOPA's Aviation USA. Frederick, Maryland: Aircraft Owners and Pilots Association, 1990.

ASA Publications
1989

FAR-AIM. Seattle, Washington: ASA Publications, Inc., 1989.

AT&T
1990

MetaTool Specification-Driven-Tool Builder User Manual.

Connes, Keith
1992

Black Box Watch. *Plane & Pilot* 28,4:27–28.

Faulk, Stuart, James Kirby, Jr.,
Skip Osborne,
D. Douglas Smith,
Steven Wartik, John Brackett,
and Paul T. Ward
1991

The Consortium Requirements Engineering Method, SPC-91140-MC. Herndon, Virginia: Software Productivity Consortium.

Heninger, Kathryn L.
1980

Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering* SE-6:2–13.

Heninger, Kathryn,
J. Kallander, David L. Parnas,
and John Shore
1978

Software Requirements for the A-7E Aircraft. Memorandum Report 3876. Washington, D.C.: Naval Research Laboratory.

Horne, Thomas A.
1989

TCAS Preview: On-Board ATC. *AOPA Pilot* 32, 6:36–40.

Nordwall, Bruce D.
1991

Foster Airdata Develops Low-Cost Collision Warning System for Navy. *Aviation Week & Space Technology* 58–59.

Parnas, David L. and Paul C.
Clements
1986

A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering* SE-12, 251–257.

Parnas, David L., Paul C.
Clements, and David M. Weiss
1985

The Modular Structure of Complex Systems. *IEEE Transactions on Software Engineering* SE-11, 259–266.

P.P. Texel & Co.
1987

On-Board Embedded Air Traffic Display/Collision Warning Monitor System ATD/CWM. Wayside, New Jersey: P.P. Texel & Co.

Ritter, Douglas S.
1992

TCAS: Boon or Boondoggle? *Aviation Safety* XII,2:1-5.

Software Productivity
Consortium
1991a

TRF2 Metaprogramming Tool User Guide, SPC-91132-MC. Herndon, Virginia: Software Productivity Consortium.

1991b

ADARTS Guidebook, SPC-91104-MC. Herndon, Virginia: Software Productivity Consortium.

1991c

Synthesis Guidebook Volume 1 Methodology Definition, SPC-91122-MC. Herndon, Virginia: Software Productivity Consortium.

1991d

Synthesis Guidebook Volume 2 Case Studies, SPC-91122-MC. Herndon, Virginia: Software Productivity Consortium.

United States Department of
Defense
1983

Reference Manual for the Ada Programming Language. ANSI/MIL-STD-1815A. United States Department of Defense, Ada Joint Program Office.

Webster
1984

Webster's II New Riverside University Dictionary. Boston, Massachusetts: The Riverside Publishing Company, 1984.