

AFRL-VA-WP-TP-2003-316

**HARDWARE-IN-THE-LOOP
SIMULATION USING OPEN
CONTROL PLATFORM**

**Stanley H. Pruett
Gary J. Slutz
Dr. James L. Paunicka
Eric Portilla**



JULY 2003

Approved for public release; distribution is unlimited.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

**AIR VEHICLES DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542**

20030822 056

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YY) July 2003		2. REPORT TYPE Conference Paper Preprint		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE HARDWARE-IN-THE-LOOP SIMULATION USING OPEN CONTROL PLATFORM				5a. CONTRACT NUMBER In-house	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER N/A	
6. AUTHOR(S) Stanley H. Pruett (AFRL/VACD) Gary J. Slutz (Protobox LLC) Dr. James L. Paunicka (Boeing) Eric Portilla (Northrop Grumman)				5d. PROJECT NUMBER N/A	
				5e. TASK NUMBER N/A	
				5f. WORK UNIT NUMBER N/A	
				8. PERFORMING ORGANIZATION REPORT NUMBER AFRL-VA-WP-TP-2003-316	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Control Simulation and Assessment Branch (AFRL/VACD) Control Sciences Division Air Vehicles Directorate Air Force Research Laboratory, Air Force Materiel Command Wright-Patterson Air Force Base, OH 45433-7542 Protobox LLC Fairborn, OH 45324-5551 ----- The Boeing Company St. Louis, MO 63166-1335 ----- Northrop Grumman El Segundo, CA 90245-2804				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/VACD	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Vehicles Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson Air Force Base, OH 45433-7542				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-VA-WP-TP-2003-316	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES To be presented at the AIAA Modeling & Simulation Technologies Conference, Austin, TX, August 11, 2003. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.					
14. ABSTRACT Air Force Research Laboratory (AFRL) researchers at the Aerospace Vehicle Technology Assessment and Simulation (AVTAS) Laboratory are currently conducting the Open Control Platform (OCP) Hardware-In-The-Loop (HITL) project sponsored by the DARPA Software Enabled Control (SEC) Program. The purpose of this project is to develop the capability to be an OCP test-bed and to evaluate the OCP controls and simulation environment for a specific test case. The OCP provides an open, middleware-enabled software framework and development platform for developers of distributed and embedded software applications. The middleware isolates the programmer from the details of the operating system and provides a mechanism for communication with other OCP software components. A Traffic Collision Avoidance System (TCAS) was chosen as a representative flight controls application to exercise OCP. The programmatic approach taken by the OCP-HITL project is a series of simulation experiments with increasing complexity.					
15. SUBJECT TERMS Open Control Platform (OCP), simulation					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 18	19a. NAME OF RESPONSIBLE PERSON (Monitor) Stanley Pruett 19b. TELEPHONE NUMBER (Include Area Code) (937) 904-6544
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			

Hardware-In-The-Loop Simulation using Open Control Platform

Stanley H. Pruett
Air Force Research Laboratory
Wright-Patterson Air Force Base, OH 45433-7505

and

Gary J. Slutz
Protobox LLC
Fairborn, OH 45324-5551

And
Dr. James L. Paunicka
The Boeing Company
St. Louis, MO 63166-1335

Eric Portilla
Northrop Grumman
El Segundo, CA 90245-2804

AIAA Modeling & Simulation Technologies Conference
August 2003, Austin, Texas

ABSTRACT

Air Force Research Laboratory (AFRL) researchers at the Aerospace Vehicle Technology Assessment and Simulation (AVTAS) Laboratory are currently conducting the Open Control Platform (OCP) Hardware-In-The-Loop (HITL) project sponsored by the DARPA Software Enabled Control (SEC) Program. The purpose of this project is to develop the capability to be an OCP test-bed and to evaluate the OCP controls and simulation environment for a specific test case. The OCP provides an open, middleware-enabled software framework and development platform for developers of distributed and embedded software applications. The middleware isolates the programmer from the details of the operating system and provides a mechanism for communication with other OCP software components. A Traffic Collision Avoidance System (TCAS) was chosen as a representative flight controls application to exercise OCP. The programmatic approach taken by the OCP-HITL project is a series of simulation experiments with increasing complexity. The first simulation was an "all software" simulation, conducted in August 2002. A portion of the "all software

simulation" was then migrated to a VME based system running VxWorks. A readiness review of the HITL simulation was conducted in March 2003 in preparation for the actual testing scheduled for the May 2003 timeframe. This readiness review tested a 2 ship non-co-operating scenario, a 2 ship cooperating scenario, and a pilot-in-the-loop scenario. Future testing is planned for formation flight and fault injection scenarios.

INTRODUCTION

The OCP is a software infrastructure being developed by the Boeing Corporation sponsored by the DARPA SEC Program. It is intended to enhance the ability to develop and test control algorithms which will eventually execute in embedded software within Uninhabited Air Vehicles (UAVs). The OCP supports distributed computing and communication allowing heterogeneous components to interoperate across platforms and network protocols while dealing with tight constraints on bandwidth, response time, and reliability. By using OCP, control engineers can concentrate on the controls problem at hand without having to worry about the details of component connectivity. The "middleware" design of OCP isolates the operating system and the underlying hardware allowing development and testing to take place on system architecture different from the final target. The

This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States

well-defined interfaces inherent with an OCP design facilitate efficient partitioned software development and insure a relatively painless final integration.

The HITL simulation using OCP was designed to evaluate some of the key OCP principles. 1. Architecture isolation/independence - The simulation incorporated three different system architectures; Window 2K on a PC, Linux on a PC, and VxWorks on a PowerPC. 2. Ease of porting - The simulation was initially built and tested as an "all software" simulation. The process initially running on the Windows 2K platform was migrated to the HITL. 3. Partitioned software development - Three teams were involved in the development of the simulation. The Boeing team provided OCP support and in particular provided key support on the integration of the PowerPC into AVTAS's architecture. The Northrop team developed all of the components associated with the Traffic Alert and Collision Avoidance System (TCAS) algorithms and the aircraft models utilized in the simulation. The AVTAS team built the OCP components necessary to interface the facilities Infinity Cube Simulator to the OCP HITL simulation.

This paper will first provide an introduction to OCP followed by a discussion on the models and the TCAS algorithms employed in the simulation. The simulation architecture and scenarios will then be presented. The paper will conclude with results of the testing to date and future plans for the program.

OCP INTRODUCTION

The OCP provides an open, middleware-enabled software framework and development platform for developers of distributed and embedded software applications. The middleware layer of the OCP provides the software layer isolating the application software from the underlying compute platform. It provides services for controlling the execution and scheduling of software components, mediating inter-component communication, and enabling distribution of application components onto a target system. The OCP includes innovative scheduling techniques, adaptive resource management, and support for dynamic reconfiguration.

The OCP is being developed by the Embedded Systems Research Team within Boeing Phantom. Assisting Boeing in these efforts are the Georgia Institute of Technology, Honeywell Labs, and the University of California, Berkeley. The OCP is

being delivered to a host of university and industrial researchers who are participating in the DARPA SEC Program.

OCP OVERVIEW AND BACKGROUND

The OCP is composed of many elements, discussed in later sections, which enable the rapid generation and test of embedded and distributed software application programs. Included among these elements are:

- 1) a middleware software infrastructure component,
 - 2) a Controls API that allows for tool-based software architecture specification and auto-coding of a software framework that implements the specification, and
 - 3) an integration with useful controls development tools, software tools, and simulation tools.
- The software infrastructure component has its heritage in the CORBA [1] (Common Object Request Broker Architecture) based, Boeing-funded software initiative called Bold Stroke. Under the Bold Stroke initiative, CORBA and its attendant object technologies were leveraged as a prime enabler for re-use of software components across product lines, and for rapid re-implementation of existing solutions on changing and evolving computing hardware and operating system platforms - primary considerations in achieving affordable avionics software development.

Boeing, in prior flight demonstrations on multiple types of aircraft, has successfully demonstrated application of this CORBA technology in the domain of non-safety critical mission processing, which required hard real time performance in tasks that ran at rates up to 40 Hz. One of the goals of OCP is to bring the advantages of object oriented programming and CORBA to the domain of UAV multi-level flight control. The application of CORBA in this domain introduces new challenges for the OCP that have spawned new requirements for OCP services. Challenges include: 1) faster rates, 2) the need to ensure vehicle stability, 3) highly accurate timing in sensor processing, and, 4) achievement of hard deadlines at flight control rates. Candidate software infrastructure services have been implemented in the OCP to meet these challenges.

OCP MIDDLEWARE

The middleware of the OCP can be used to fuse embedded and distributed system application software components together, controlling their execution and communication.

A primary motivating factor in implementing a middleware-based architecture was the promise of isolating the application components from the underlying platforms. This allowed for a more cost-effective path for implementing common software components that could be used (1) across different product lines, and (2) could be rehosted onto evolving embedded computing platforms. Support for rapid re-implementation of existing, tested designs onto evolving computing platforms is important for maintaining an effectiveness advantage in currently fielded embedded systems.

These evolving computing platforms are starting to be dominated by commercial hardware and software components, which have more dynamic life-cycles than previous military components, and which create more opportunities for incorporation of computing advances into existing systems. Further, more commercial and military flying platforms are experiencing longer lifetimes, and the ability to inexpensively re-host existing functional and tested software on updated computing platforms is very attractive.

The embedded software framework of the OCP inherits the RT-CORBA (Real Time-CORBA) based middleware of Bold Stroke. The inherent advantages of a middleware-based solution within OCP should prove to be an enabler for current and future embedded and distributed software developments.

Use of the CORBA-based middleware helps isolate application software components from the underlying hardware and operating systems in the computing platform. The resulting layered architecture is illustrated in Figure 1.

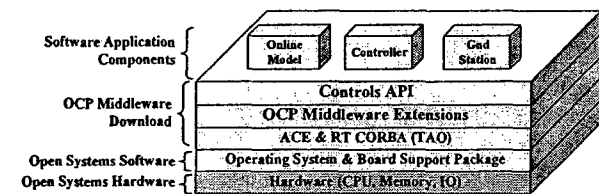


Figure 1. OCP Layered Architecture

The middleware also facilitates distributed processing and inter-component communications by supporting CORBA event-based communication, as illustrated in Figure 2.

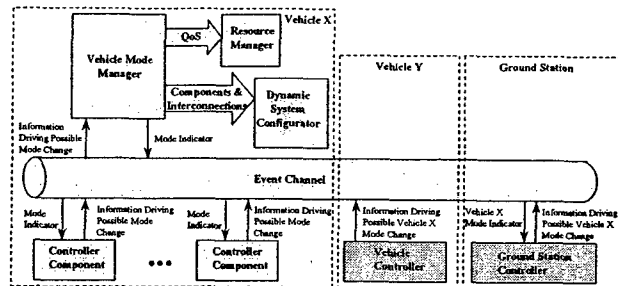


Figure 2. Distributed Processing and Inter-Component Communications

The ability of the OCP to isolate application components from the underlying system also can be used to enable efficient embedded and distributed system software development. For initial design and development, application code integrated with the OCP middleware can be executed on familiar desktop systems, such as those using Windows or Linux operating systems. Certain levels of software testing can take place more conveniently on the desktop. Then, the operating system and hardware isolation features of the middleware would be instrumental in implementing the smooth transition of the developed embedded application code to the embedded hardware target – perhaps a target executing a POSIX-compliant Real-Time Operating System (RTOS), such as VxWorks or QNX. Isolation of the application would result in minimal code changes during transition to the embedded target.

The middleware of OCP is being designed so that it offers the embedded software developer a rich set of features desirable for real-time applications. These features include dynamic scheduling; adaptive resource management; dynamic reconfiguration; hybrid system mode switching support; and convenient access to real-time triggers.

Embedded applications built with the OCP middleware exhibit a layered architecture, as shown in Figure 3. The bottom software layer shown is that of the operating system (OS). The OCP has been designed to allow applications to be executable on a variety of OSs, including desktop non-real time Windows and Linux, as well as RTOSs, such as VxWorks and QNX.

The OS portability is enabled with the next highest layer, an OS-abstraction layer implemented by the open-source Adaptive Communications Environment (ACE) software. ACE provides common OS services to software residing in the layers above it. Software in these upper layers can access useful

OS services with ACE calls, and ACE then translates the requests into OS-specific requests.

The advantages of CORBA are made accessible in OCP by the next highest layer, the TAO (The ACE ORB) layer. TAO and ACE are available as open source implementations from Washington University in St. Louis. TAO has been developed as an ORB (Object Request Broker) implementation that is portable across a variety of underlying compute platforms. TAO itself provides some real-time performance extensions to CORBA.

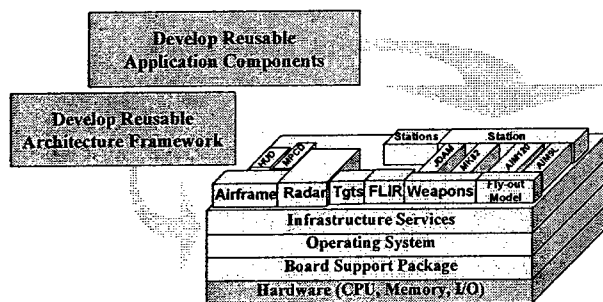


Figure 3. Layered Architecture of an OCP-Based Application

Above the TAO layer, the OCP middleware presents to the application a rich set of run-time services, many of them based on standard CORBA. On top of ACE and TAO, some real-time performance extensions were developed by Boeing, under the Bold Stroke project, and utilized in fighter aircraft Mission Management embedded software. The OCP includes further real-time performance support to handle the requirements of UAV control. Some of the more important services are discussed below.

CORBA Services (Real-Time Publish / Subscribe)

The CORBA event channel (EC) can be used to route data between software components without resorting to parameter passing or global memory pools, both of which are difficult to design and maintain for applications distributed over multiple threads, processes, and processors. Data being generated by a software component, if it is needed elsewhere in a system, can be published over the EC. Other software components in the system which have access to the ORB, and which need a particular input, can subscribe through the EC for that input. The software component publishing the data (as an output) and the one or more software

components subscribing to that data (as an input) can reside anywhere the ORB exists -- in the same process, in different processes within the same computer, on different computers within the same chassis, on different computers at different locations (e.g., two different vehicles, or a vehicle and a ground station), etc.

The CORBA EC can also be used to trigger the execution of a software component based on profiles of arriving inputs. This can be used to relieve the software designer from trying to derive a correct cyclic executive that hopefully satisfies all input data dependencies correctly, which can be a daunting task for applications that have large numbers of application software components, have multiple threads, which span more than one process, and which span more than one processor.

OCP Services (Resource Management)

The OCP's Resource Management component provides a mechanism for controlling and making better use of compute resources in the executing system. The OCP resource management component is an extension of the Honeywell Labs Real Time Adaptive Resource Management (RT-ARM) capability. [2] With RT-ARM, it is possible to specify quality of service (QoS) information about the various software components in the system. An example QoS specification would be the allowable rates of execution for the proper operation of a software component. While the system is in operation, the RT-ARM functionality can adapt the scheduling behavior of the system to optimize utilization of the finite embedded compute resources based on the software components which are currently active and their QoS information. The resulting execution rates of the components, as scheduled by the RT-ARM capability, are communicated to the individual software components so that they can then modify their behavior based on their assigned rate (e.g., modify controller gains). RT-ARM makes use of the TAO scheduling component [3].

OCP Services (Hybrid Systems Mode Change Support -- the Transition Service)

A hybrid system combines both continuous and discrete elements. For example, in a typical flight controls system, the lower levels of the architecture tend to be designed as continuous time controllers. When moving to higher levels of the architecture, controllers tend to be of the discrete supervisory type. The composite system can

function like a pseudo-continuous controller able to operate in one or more distinct modes.

To help meet the needs of this hybrid system philosophy, system mode support has been added to the OCP with the Transition Service. Each system mode can be characterized as being made up of a specific profile of active (and inactive) software components, a specific QoS profile for the active components, and a specific profile of input/output interconnections between active components. The Transition Service allows components to identify the current mode of the system, allows components to trigger mode changes, and allows for smooth mode transitions at appropriate times of system operation (e.g., after inner-loop control actuator command writes).

Note also that for each system mode, a different profile of QoS parameters can be specified for a software component. This provides a powerful way to allow use of RT-ARM to make the best use of available on-board resources, since some components are more critical in some system modes and would therefore have their more challenging resource demands reflected in their QoS specification for those modes.

OCP Services (Accurate Time Triggering -- the Timer Service)

UAV flight control applications have stringent real-time constraints on operation. For example, flight control sensor sampling must be accomplished at precise time intervals, with very little frame-to-frame jitter and without missing a sample. The Timer Service in the OCP provides a way to accurately trigger a software component based on time. This is accomplished by providing a convenient API linking a physical real-time clock on the embedded processor board to a software component execution trigger.

OCP Services (Performance Optimizations)

Performance optimizations are being implemented in the OCP to reduce the amount of time that an OCP-based application spends in the CORBA-based middleware layer. These optimizations include use of light-weight EC events, client-side caching to reduce the need for data passing in a distributed application, and the ability to allow efficient protocols to be plugged into the ORB to optimize data transfer speeds between specific components.

OCP CONTROLS API

As mentioned previously, the OCP provides several advanced mechanisms such as adaptive resource management, reconfiguration during mode switches, and access to highly accurate timing sources for component triggering. To help hide the complexity of these features from the controls designer, and to shield the controls designer from C++ or object oriented programming, the OCP includes the Controls API -- a controls designer abstraction layer above the RT CORBA implementation. This API allows the designer to focus on familiar tools and terminology whilst enabling the use of RT CORBA extensions. This abstract layer was a collaborative Boeing - Georgia Tech effort. The Controls API provides an interface for managing OCP components, setting system information, and controlling system execution.

The Controls API allows a software system designer to lay out the system graphically using the popular Simulink tool. Here, the designer specifies the software component names, their inputs and outputs, and the interconnections with inputs and outputs of other components.

This initial layout is then decorated with additional information by the system designer using another graphical tool written in Java. Here, all other pertinent information about the software system is specified. This other information includes specification of system modes, the active software components and their interconnections in each mode, QoS information for active components in each mode, specifications of which components get triggered by highly accurate timers, allocations of components to different processes, etc. The completed system model is then sent to the final portion of the Controls API for automatic generation of a C++ framework that will trigger software components based on the system model, that will route inputs and outputs based on the model, and that will allow system mode changes as specified in the model. The autogenerated framework contains placeholders in the code for the controls designer to insert the code for the individual software components.

OCP INTEGRATION WITH CONTROLS DEVELOPMENT TOOLS AND SIMULATION TOOLS

The OCP provides integration with popular and useful controls and software design, development, and testing tools. Commercial tools like Microsoft Visual C++, Microsoft Visual Debugger, the VxWorks real-time operating system, and Matlab/Simulink have wide acceptance in the software and controls communities. Buildable and runnable

examples delivered with the OCP to illustrate OCP features make use of these well-supported commercial products. The debugger can be used, for example, during a running vehicle simulation to set breakpoints, single-step, and perform other useful testing functions while simulation displays show the dynamic state of the vehicle.

TCAS OVERVIEW

Over the course of the HITL program, there has been a growing emphasis on automated collision avoidance. As a result, logic from the commercially packaged software, TCAS algorithm, was used as a base model for the design of a mission manager. The basic TCAS algorithm consists of two main components. The first component detects and alerts pilots to possible vehicle conflicts, while the second computes an advised rate of climb to remove the threat. Since the vehicle being used in the HITL simulation is an unmanned vehicle, the pilot advisory is converted into a 4-D waypoint command which drives a "fly to point within time" outer loop guidance routine.

NON-CO-OPERATING TCAS

The TCAS detection algorithm monitors the inertial position and velocity of intruder vehicles in the surrounding area. Using this data it calculates whether any pose a threat of collision or near miss (based on a user specified separation in both the horizontal and vertical planes. TCAS then uses an estimated time to the Closest Point of Approach (CPA) as a guideline for prioritizing threats. A time variable concept, simplified tau (τ_u), was developed to give an approximation of the time to CPA by dividing slant range by the closing speed (range/range rate) in the xy-plane. The tau time constant is used to ensure that there is ample time to maneuver out of danger,

$$\tau_u = -\frac{r}{\dot{r}}$$

Equation 1 – Simplified tau time constant

The threshold value for τ_u is predetermined based on the maneuverability of the vehicle. When the calculated value drops below this threshold the intruder is considered a threat in the horizontal plane and is evaluated to see if it also breaches the vertical separation threshold. The simplified tau evaluation only sets a threshold for time which cause problems when detecting intruders with slow closure rates. If the relative horizontal closure rate is slow enough, an intruder vehicle can get well within the desired minimum distance separa-

tion before it is recognized as a threat. In order to include time and distance constraints, a distance threshold (D_m) constant is added to the simplified tau equation giving the Bramson Tau Criteria (τ_b), which protects the vehicle from failure due to slow closure rates.

$$\tau_b = -\frac{r}{\dot{r}} + \frac{D_m^2}{r\dot{r}}$$

Equation 2 Bramson tau time constant

Once it has been established that an intruder will breach the horizontally protected space, TCAS uses both tau values (τ_u , τ_b) to determine if the intruder will also breach the vertical separation desired. Due to the fact that altitude data from on board transponders are in discrete altitude steps of 100 feet, TCAS creates a critical area to evaluate whether the intruder will breach the vertical safety area. Two vertical separation estimates are calculated using each of the tau time constants:

$$m_{v1} = (z_o - z_i) + \tau_u (\dot{z}_o - \dot{z}_i)$$

$$m_{v2} = (z_o - z_i) + \tau_b (\dot{z}_o - \dot{z}_i)$$

Equations 3 & 4 - Vertical separation estimates

z_o, z_i = altitude of own vehicle and intruder

$\dot{z}_o, \dot{z}_i$ = rate of climb of own vehicle and intruder

Once the calculated separation value of m_{v1} or m_{v2} dips below the user specified threshold, the intruder is considered a collision threat. Each intruder that breaks this threshold is arranged in order of their corresponding tau from least to greatest and then sent to the avoidance algorithm for de-confliction. A sequence diagram of the detection process is shown in figure 4.

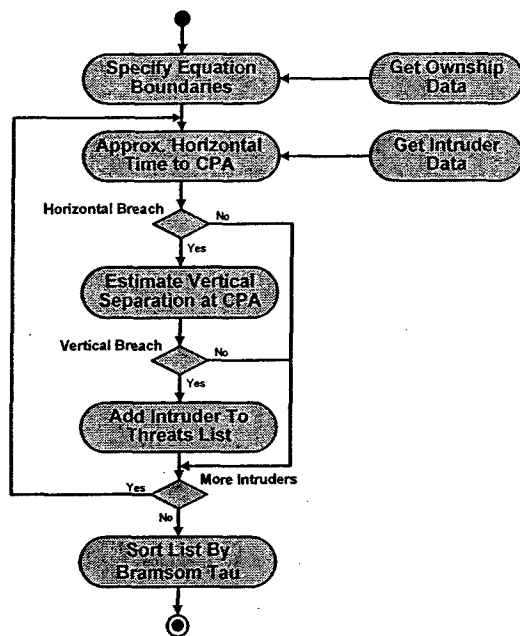


Figure 4 - TCAS Detection Sequence Diagram

The list of intruders is then sent to the avoidance algorithm where it is used to map out multiple paths using a branching node method. User predefined possible rates of climb are used to create these paths. The first node is designated as the position of the ownship vehicle, while each set of following nodes are set by the next smallest tau until all intruders are considered. The number of total possible paths is based on the number of predetermined rates of climb along with the number of intruders:

$$\# \text{ paths} = \# \text{ rates of climb}^{\# \text{ intruders}}$$

Equation 5 - Total Possible Paths

A cost algorithm is then used to evaluate which path causes the ownship vehicle to deviate the least from its current path while still achieving the desired difference in altitudes between the vehicles. Each node has a cost associated with it determined by the estimated separation distance from the intruder at CPA. The farther from the intruder the node is, the greater the cost. This is true unless the node breaks the minimum clearance threshold, in which case the node is given an extremely high cost essentially removing it and all of its sub paths from the list of possible avoidance maneuvers. Once each node's cost has been evaluated, the overall cost of each path is calculated by summing the node costs, which it passes through. The path with the least cost is the one chosen by TCAS.

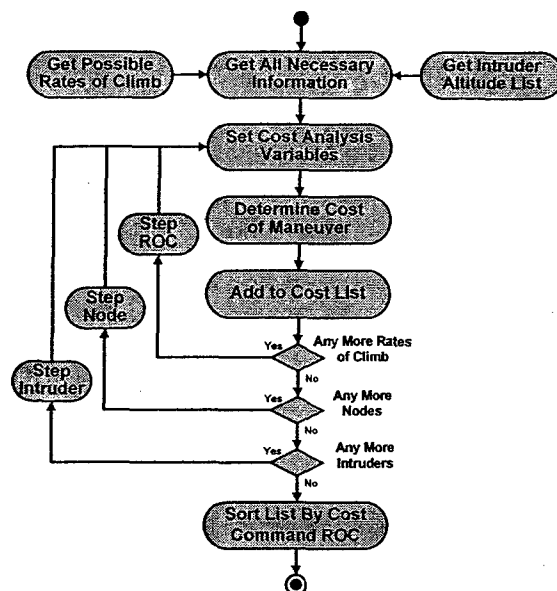


Figure 5 - TCAS Avoidance Sequence Diagram

The branching node method is illustrated in Figure 6, which depicts the avoidance algorithm given three possible rates of climb and three intruders breaching the minimum clearance distance. There are a total of 27 possible paths for the vehicle to evaluate. The circle surrounding each intruder is the defined bubble of safe separation. All 17 of the paths, which pass through these bubbles, are assigned a high cost and are represented by the dashed lines. Of the remaining 10 paths, the bold path is determined to be the least expensive.

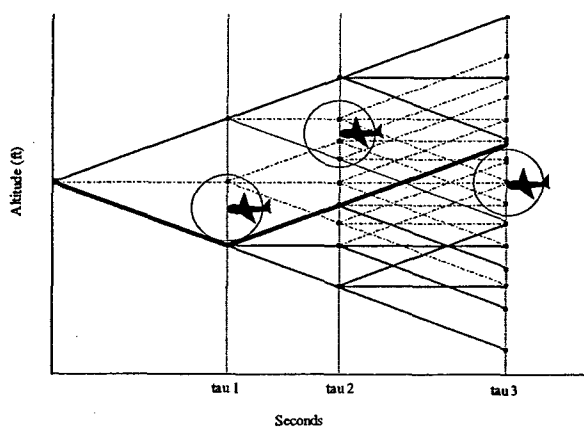


Figure 6 - TCAS Avoidance Example

Since the tau variable is based on time to CPA and not actual distances, the protected volume of each incidence varies based on the speeds and headings of the aircraft involved. These are the limiting factors of TCAS. The maximum rate of closure that it can protect against is 1200 knots horizontally and 10,000 ft/min vertically.

Since the original intention of TCAS is to be an advisory program, there were modifications necessary to convert it to a mission manager. The first change needed was to output 4-D waypoints to drive the outer loop guidance routine, instead of an advisory rate of climb. To accomplish this, the mission manager uses the commanded rate of climb combined with the time to CPA calculated by TCAS to estimate the altitude that the vehicle must achieve to avoid the threat. This altitude, along with the mission targets x, y positions, provide the point to fly to for the guidance routine and the mission targets Estimated Time of Arrival (ETA) filled in the time used to control the speed. During these initial testing, the TCAS algorithm would turn on and off multiple times as the vehicle change its rate of climb in an attempt to obtain the commanded altitude. The results were a degradation in the performance of the collision avoidance. To reduce the frequency of this cycling, two control timers were added to the system. The first timer holds the initial TCAS command once an evasive maneuver is commanded. This allows time for the vehicle to respond to the request before re-evaluating the situation. The second timer defines the length of time that the command is held after the threat has past before TCAS can turn off. The top-level logic of the overall mission manager including these modifications are displayed in Figure 7. The detection and avoidance sequence diagrams are represented by the blocks titled "Detect" and "Avoid" respectively. The two added timers are referred to as the "On Timer" and the "Hold Timer".

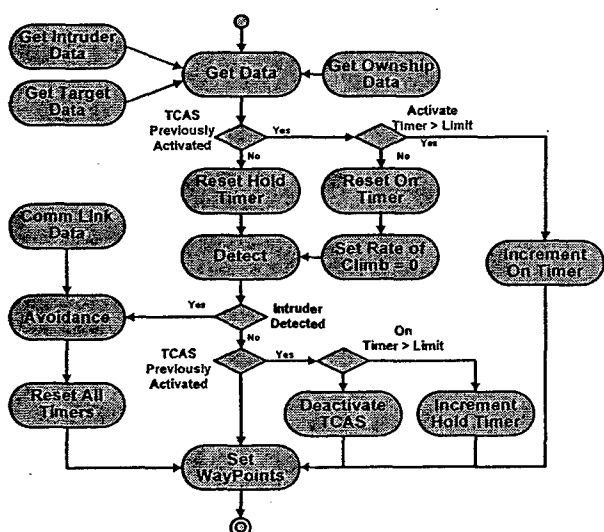


Figure 7 – Modified TCAS Top-Level Sequence Diagram

CO-OPERATING TCAS

In Scenario 2 of the HITL program, there arose the need to perform a coordinated collision avoidance maneuver. To achieve vehicle negotiation, the cost function internal to the TCAS algorithm was modified to allow communicated data to be used during its evaluation. The cost function was modified to consider three factors. The first being the available vehicle states. The cost function does a preliminary evaluation of altitude and rate of climb of each vehicle and flags a recommended command of either climb or dive. For ambiguous cases (i.e. vehicles at common altitude and rate of climb) a "right of way" system was designed to use the vehicles longitudinal and lateral coordinates to resolve the issue. This removes the chance of two vehicles initially deciding to perform the same maneuver. This recommendation is then used to add a gain to the originally calculated cost of all the rates of climb, which contradict it. The gain to the function effectively raises the cost of the maneuver removing the likelihood of being chosen while still leaving the option available if all else fails. Next, the cost function evaluates the communication data being sent between the vehicles. This communication is specified to be a signal whether or not the other vehicle's TCAS is actively commanding a collision avoidance maneuver and a signal stating what that commanded altitude is. With this information the cost algorithm calculates a better estimate on what maneuver is required to achieve the separation desired. The last modification to the cost function was the ability to set altitude constraints to avoid the vehicles being commanded above their physical flight ceiling or driven into the ground, which was a problem encountered during early tests.

MODELS

All the scenarios tested in the HITL program use two base vehicle models. The ownship, which is run HITL, is a UAV type vehicle while the intruder is an entirely software model of a generic fighter class vehicle. Each vehicle uses a generic outer loop controller which acts as an autopilot consisting of the TCAS mission manager described earlier.

The introduction of the pilot to the simulation brought many new issues to the table. The first and most notable was the fact that when the pilot disengaged the autopilot, the model and inner loop became unstable. The outer loop was able to hide these problems but the pilot-in-the-loop made these deficiencies very apparent. Another prob-

lem, which arose from the pilot testing, was the case of severe maneuvering causing the vehicle to slip off of the aero data tables used by the model causing the simulation to crash. These problems, which otherwise would have gone unnoticed, were fixed resulting in a much more robust model for future use on HITL and other programs.

TEST SCENARIOS

Each vehicle in the HITL simulation is broken into eleven separate OCP Components (Figure 8). These eleven components are then separated into processes based on the hardware architecture. Since the UAV is the only vehicle to be flown HITL, it is the only one that is separated into two processes. One process runs the controller on the hardware and another updates the model outside of the hardware on a WinNT machine. All other models are software only and therefore run on a single process. This structure is used as the basis for the HITL programs five specified software test scenarios. Each of these scenarios is then executed under multiple collision course approach angles (specifically acute and obtuse). By running the scenario under acute and obtuse relative heading conditions, the TCAS algorithm can be tested under its worst-cases: the extremes of high and low closure rates. The case of low rate closure causes the intruder vehicle to breach the separation threshold desired before the tau criteria, is breached and the obtuse case causes the closure rate to test the upper limits of TCAS.

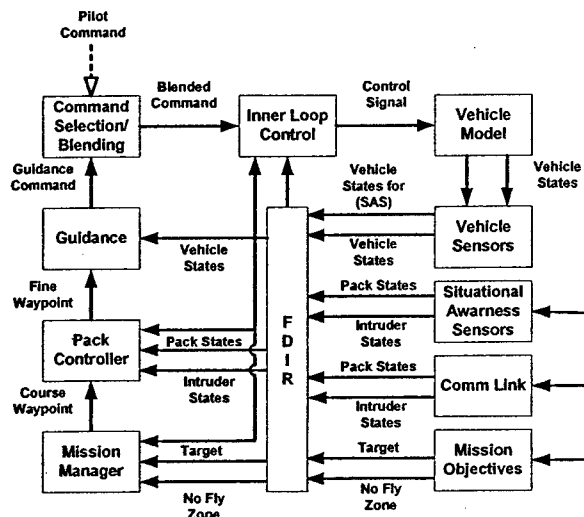


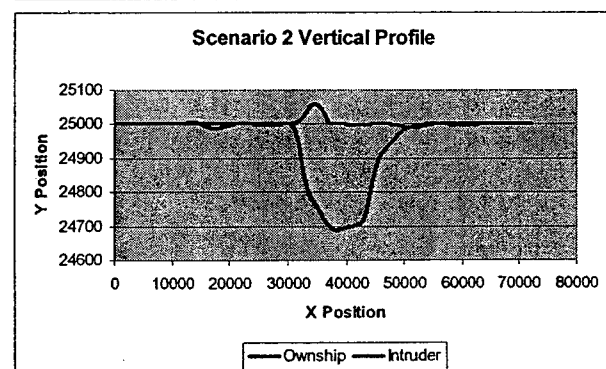
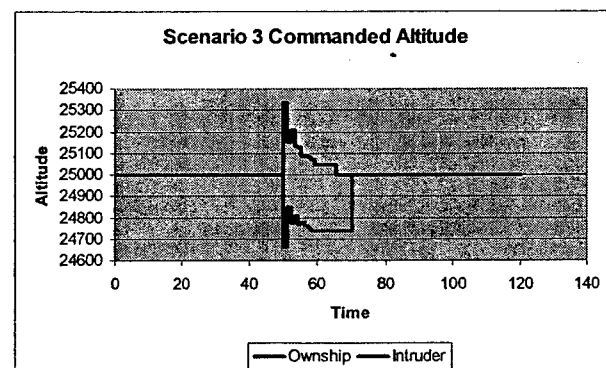
Figure 8 - Block Diagram of OCP Components

TEST RESULTS

The first scenario successfully tested two vehicles traveling on a collision course and the ownship performs an evasive climb or dive to avoid the in-

truder. In this scenario there is no communication between the vehicles and it is assumed that the intruder is blind to what is happening.

In the second scenario tested, both vehicles are equipped with the TCAS mission manager. The initial collision courses are the same as the first scenario but instead of the ownship taking on the full responsibility of the evasive maneuver, both vehicles react to the possible collision. The communication between the two vehicles, which was not present in the first scenario, allows the vehicles to negotiate a coordinated de-confliction plan where one dives and the other climbs. It is noted that for the initial instance in which the vehicles detect a collision, each vehicle attempts to perform the entire de-confliction alone. Once communications of the detection between the vehicles begin and they negotiate a solution, each vehicle commands half of the change in altitude required to accomplish the desired separation. This communication continues throughout the maneuver and is updated constantly resulting in the more maneuverable vehicle being requested to do more of the change in altitude. This is seen in the time histories where the more agile generic fighter aircraft is asked to perform more of the maneuver, as time goes on based on its performance response.



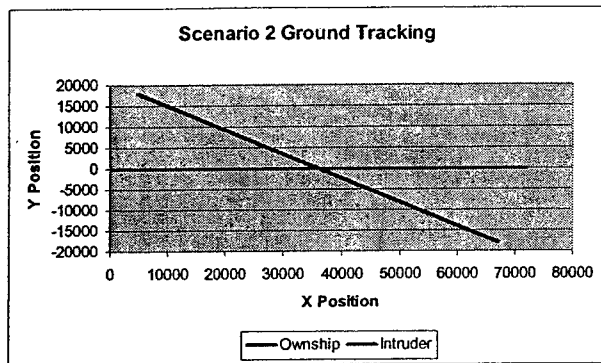


Figure 9 – Scenario 2 Acute Time Histories

Scenario 3 adds a human-in-the-loop aspect to scenario 2 which allowed robustness testing of the mission manager. (Scenario 3 is discussed in more detail in the next section) The insertion of the pilot allows us to examine the reaction of the mission manager to unanticipated decisions by the other vehicle. The ownship has to react to an intruder pilot overriding the negotiated TCAS outer loop command.

The final two scenarios to be completed are adding a wingman to the ownship and fault injection. The wingman will perform basic station keeping abilities while only communicating with the ownship. The ownship will be the intermediary between the intruder and the wingman, who will not directly communicate throughout the scenario. The final scenario is fault injection into the system. This scenario will test the fault detection algorithm being developed as well as the reconfiguring of the vehicle's performance. The fault injector will be another component added to the architecture, which will insert faults into the model as data is passed to the fault detection isolation component.

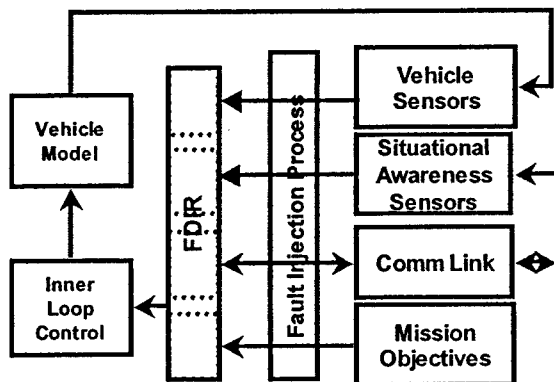


Figure 10 – Fault Detection Architecture

Pilot-in-the-Loop HITL simulation

The pilot-in-the-loop portion of the simulation (scenario 3) is designed to evaluate the process of integrating an OCP based simulation into the AV-TAS simulation architecture and to prepare AV-TAS as a future OCP test bed.

The infinity cube simulator is a state-of-the-art fix based research simulator containing four collimated displays in a cube arrangement, a projected Heads Up Display (HUD), and a 29" monitor for the Heads Down Display (HDD). It contains a stick and throttle and a set of generic rudder pedals. The out the window (OTW) display is driven by Subscene, a locally developed image generation software package that runs on PC's under Linux. Figure 11 shows the architecture employed for the pilot-in-the-loop simulation from the OCP vantage point

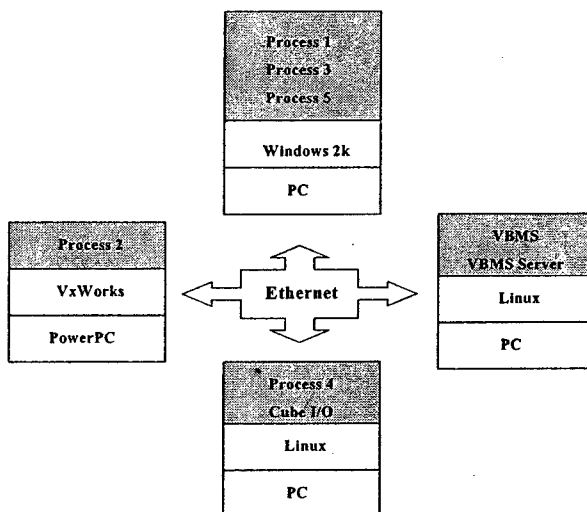


Figure 11 - Pilot-In-Loop-Architecture

A Windows 2k platform runs OCP processes 1, 3, and 5. Process 1 is the ownship aircraft model. Process 3 contains both the aircraft model and the controller for the intruder. Process 5 communicates with a Virtual Battle space Management System (VBMS) server to update VBMS in real-time. One Linux platform is used to host the VBMS display and an associated CORBA based VBMS server. A second Linux platform runs OCP process 4 and the infinity cube I/O process. Process 4 interfaces the rest of the OCP simulation to the infinity cube. This platform will be discussed in more detail in succeeding paragraphs. The final platform is the HITL system and consists of a PowerPC running VxWorks. The controller process for the ownship (Process 2) is run on this platform. All machines are connected to a dedicated Ethernet network.

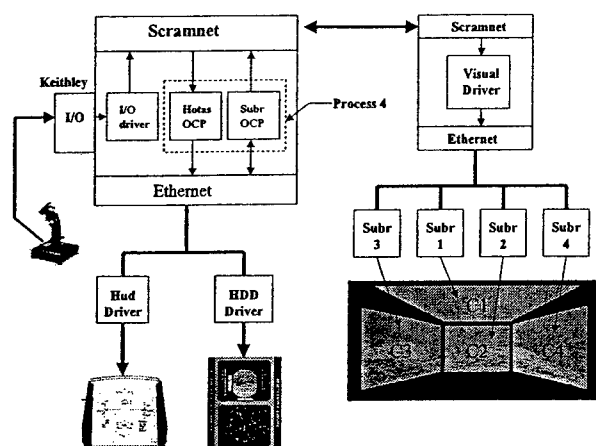


Figure 12 – Infinity Cube Interface

Figure 12 shows in more detail how the OCP portion of the simulation is interfaced to the infinity cube.

Process 4 consists of two OCP components. The Hands On Stick And Throttle (HOTAS) component is responsible for providing the intruder model with the stick, throttle, and rudder pedal inputs from the infinity cube. The analog and digital signals from the stick, rudder pedals, and the throttle are read via a Keithley I/O card by the I/O process. This process then places the data into SCRAMNet memory where it is read by the HOTAS OCP component and sent to the intruder model via standard OCP signaling mechanisms. The Subscene component receives ownship and intruder state information via OCP signaling mechanisms and writes this information into SCRAMNet. A Visual Driver process running on another Linux platform reads the state information from SCRAMNet and sends it to Subscene via Ethernet. Subscene is run across 4 Linux based platforms with one platform per infinity cube channel. The Subscene OCP component also sends the state information to two other Linux platforms over Ethernet. These platforms generate the symbology for the HUD and the HDD.

A standard HUD is being used with a modification to annunciate the condition when TCAS detection occurs. An existing HDD was modified to include a simple vertical and horizontal situation display to facilitate re-acquiring the ownship aircraft. The intruder model is set up to blend the inputs from the stick and the autopilot for stick inputs below a certain threshold. When the stick inputs exceeded this threshold the autopilot is disengaged. The

paddle switch at the base of the stick is used to re-engage the autopilot.

The general guidelines for running scenario 3 with pilot-in-the-loop are to allow the initial avoidance maneuvers to take place and then try to re-acquire the ownship and force another collision avoidance situation. Several team members flew the simulation in preparation for the readiness review. This initial testing showed that all of the interfaces were functioning properly. The cube display system provided a compelling view of the initial avoidance maneuvers performed by TCAS. Stick and throttle inputs were verified to be correct though some difficulty was experienced in flying the intruder manually due to the aero model currently being employed. During this initial checkout the ownship model was successfully re-acquired on several occasions and performed evasive maneuvers as expected.

CONCLUSIONS

The HITL simulation utilizing the OCP is showing promising results at this point in the project. Software developed by three different teams is being integrated with little difficulty due to the well-defined OCP interfaces. The ownship controller process that was developed and tested on a Win2k machine was easily ported to a VxWorks machine. The current HITL simulation is successfully running on a heterogeneous network of machines.

Testing of the 2-ship non-co-operating, the 2 ship co-operating, and the pilot-in-the-loop scenarios is scheduled for May 2003. Testing of the formation flight and fault injection scenarios is scheduled for late summer of 2003.

REFERENCES

1. M. Henning and S. Vinoski, Advanced CORBA Programming With C++, Addison-Wesley Longman, 1999.
2. "On Adaptive Resource Allocation for Complex Real-Time Applications", Rosu, D., Schwan, K., Yalmanchili, S., Jha, R., Proceedings of the IEEE Real-Time Systems Symposium, December 1997.
3. "Adaptive Scheduling for Real-time, Embedded Information Systems", Bryan S. Doerr, Thomas Venturella, Rakesh Jha, Christopher D. Gill, and Douglas C. Schmidt, Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC), St. Louis, Missouri, October 24-29, 1999.