ABSTRACT

The paper describes the Airbase Logistics System architecture and the methodology and issues we addressed to incorporate a legacy constructive simulation into a Distributed Interactive Simulation protocols compatible distributed virtual environment. We also describe the communication architecture developed for integrating the Airbase Logistics System, a notional wing operations center, and an associated 3D graphical model into a consistent, seamless portrayal of the airbase and its operational status to other distributed virtual environment participants. We discuss issues related to maintaining a consistent description of airbase status between the 3D graphical model and the Airbase Logistics System.

1. INTRODUCTION

An abiding shortfall in the fidelity of distributed virtual environments (DVEs) is the lack of an accurate portrayal of airbase logistics [1]. As a result, the DVE does not accurately portray the difficulties inherent within the real world and DVE participants can develop unrealistic expectations about the ease of real world operations. The logistical representation shortfall does not result from a lack of awareness of the need for logistics modeling in DVEs but rather results from the cost of acquiring data to produce a valid model and the cost of assembling a computational model based upon the data. Our research was undertaken to remedy this modeling deficiency for airbase logistics. Because of cost, we decided to incorporate an existing constructive airbase logistics model, CWTSAR, into Distributed Interactive Simulation (DIS) based DVEs. This DIS-compatible system is called the Airbase Logistics System (ALS).

The Airbase Logistics System is composed of CWTSAR, an object-oriented run-time data repository (the Common Object DataBase (CODB)), and ModSAF network utilities. CWTSAR computes the status of the airbase based upon information about the base and the distributed environment stored in the CODB. The CODB is the interface between CWTSAR and ModSAF. The real-time inputs to CWTSAR come from Protocol Data Units (PDUs) and from formatted text messages. The airbase’s state is recomputed at 15 minute time intervals or upon receipt of a message that contains significant changes in status for the airbase or any of its components.

ALS, as a DVE entity, provides real-time output data for a number of important components of airbase state. These components include sortie launch, airbase status, causes of maintenance shortfalls, and number of aircraft on missions and on base. Furthermore, ALS maintains individual aircraft status, location and configuration of aircraft on base, numbers of aircraft lost or damaged, number of mission capable aircraft, available munitions by type, and petroleum, oil and lubricants (POL) availability.

The next section presents background information for our research. Section Three details the requirements we developed for the ALS. Section Four presents the architecture and design of the ALS. Section Five contains our conclusions and recommendations for future work.

2. BACKGROUND

The ALS is designed to operate within DIS-based distributed virtual environments. A distributed virtual environment (DVE) is a large-scale, networked, computer-based, virtual world wherein several thousand human and computer controlled entities can interact [1]. DVE technology permits real-time human interaction with complex environments based upon 3D graphical descriptions that interact and move using models of real world physical constraints. Therefore, the DVE requires models of the terrain and other features such as roads, forests, rivers, bridges and buildings. A DVE uses long-haul or local networked virtual environment hosts to form a single environment wherein each host has its own local model of the environment. The hosts, or network nodes, broadcast the significant changes in the environment’s state.
to all the other hosts. The broadcasts allow the participants to interact at a distance and to maintain local models of the environment that satisfy the fidelity criteria for each participating entity. The ALS is designed to operate within a DVE and to provide a realistic model of aircraft sortie generation for planners.

The core of the ALS logistics modeling capability is CWTSAR. CWTSAR is a constructive airbase logistics model written in FORTRAN that was developed by the USAF Armstrong Laboratory over a period of ten years starting in the late 1970s. CWTSAR models an airbase, its logistics, and the resulting sortie generation capability during normal airbase operation and post-attack. The CWTSAR model of the airbase considers factors such as POL availability, munitions availability, and base personnel availability in sortie generation modeling throughout the duration of the simulation. Aircraft type, mission type, weapons load, flight size, abort status, airbase status, numbers of available aircraft per mission, munitions availability, POL, and airbase facilities status are all maintained within the CWTSAR model. CWTSAR was designed to operate as a standalone simulation system, and therefore it has no facilities for communicating with external simulations. Furthermore, it was designed as a complete simulation system for airbase operations, so it includes facilities for determining level of damage suffered during an attack, functions for tracking available personnel, and the means to simulate multiple airbases within a single execution. CWTSAR can also model an attack on its airbase using Monte Carlo methods, determine where damage has occurred, and model the time required to repair the damage. CWTSAR does not model individuals or individual pieces of equipment, it deals with them in the aggregate. However, aircraft, airbase facilities, and individual bomb craters are modeled individually.

The ALS’ architecture is based upon the Common Object DataBase (CODB) [2]. The Common Object DataBase is a data-handling architecture. It uses structured classes, data containers, and a central runtime data repository to transmit data between objects. This architecture reduces the coupling in a simulation by reducing the amount of information that a class must maintain about other classes. As a result, the simulation engineer can abstract away from the classes and methods that produce data. To acquire the data the simulation engineer must only access the container in the CODB where the information resides.

In the ALS, the CODB provides an inheritable, reusable interface to the computer hardware, input and output devices. When ALS begins execution, the CODB loads the description of the static portions of the distributed virtual environment. The CODB also loads the 3D models for all the actors in the DVE and the initialization data for CWTSAR.

The ALS is intended to operate within DVEs that employ the distributed interactive simulation (DIS) suite of protocols, IEEE 1278-1993, to communicate between the computer hosts that support the DVE. DIS is a time and space coherent representation of a virtual environment. DIS operates in the human perception time frame, at or above the 100ms level. This means that simulated events requiring fine-grained synchronization, such as radar emissions, must be handled outside of the DIS protocols. DIS is designed to achieve interactive representations of virtual environments by the interconnection of individual simulators, or hosts, at distributed locations. Each host must determine what is actually perceived of its entities and only communicates changes in the state of entities for which it is responsible.

In the DIS paradigm, the DVE contains actors and entities that interact through the asynchronous broadcast of event and state information. Examples of event and state information are significant changes in actor velocity and acceleration, simulated radio broadcasts, weather state, radar broadcasts, weapons fire, patient heartbeat, human arm posture, or on-board fuel status. The DIS protocols encode and transport this information about the actors and events between the participating hosts. To maintain compatibility and interoperability between the hosts, the host computers must have access to the same, or at least equivalent, representation of the environment.

Distributed interactive simulation can support large-scale virtual environments because it allows a wide variety of actors and other entities to interact in real time. The classes of entities are: 1) autonomous, real-time computer-generated actors (also called semi-autonomous forces); 2) phenomenology entities (clouds, air density, moisture, precipitation, wind speed), radar emissions, radio emissions, or infrared emissions); 3) actors controlled by humans; 4) actors that are avatars for systems operating in the real, physical world; and 5) actors controlled by traditional computer-based simulation systems.

The DIS approach to distributed simulation requires participating sites to meet several requirements. These requirements are: 1) autonomous operation; 2) object and event based simulation; 3) ground truth and state change information broadcast; and 4) dead reckoning capability. The use of autonomous (or self-contained) hosts, allows each site to operate independently and eliminates a central system clock or event server and facilitates the interconnection of existing simulators. The object and event based simulation approach limits inter-host communication to the transmission of activities related to entities and entity-related events that occur within the DVE. Ground truth and state change information broadcast requires hosts to broadcast accurate data and to broadcast when an entity’s state significantly changes.

The ALS’ DIS interface is ModSAF. ModSAF provides utilities for broadcasting aircraft and weapons
entity state PDUs and for receiving entity state PDUs. ModSAF is a set of software modules, C libraries, that can be used to construct DIS applications. ModSAF also provides an interface to the Command & Control Simulation Interface Language (CCSIL).

CCSIL is a communications protocol for use between entities within a DIS-based DVE. CCSIL includes a set of standard message formats and a vocabulary of terms that can be used to fill in the messages. CCSIL can be used to simulate the messages that are transmitted during air to air, air to ground, close air support, defensive counter air, and sweep missions. CCSIL can also be used to transfer other standard messages such as mission status, airspace status, flight check in, air tasking order, contact report, base incident report, and battle damage assessment report. CCSIL uses real-world message formats for these message types, which increases the fidelity of the content and format of the data that is transmitted. For example, when airbase status is being reported, CCSIL is able to support the transfer of data reporting the airbase name, status code, time of report, aircraft types, aircraft quantity, aircraft status, fuel type, oil type, and days of supply on hand in the same manner and format as in the real world.

One challenge with any simulation architecture is maintaining separation and minimizing coupling between the major application components. The solution we adopted involves using containers to move data en masse from/to a component to/from the CODB. We structure the containers so that each type as well as each sub-type for each type in the container has a dedicated, pre-allocated portion of the container. We allocate an identical amount of space for each type, even though in some instances this approach leaves some data space unused for an individual type’s slot in the container. In addition, some space in the container may not be used because the container and sub-container sizes are fixed for a specific number of types when the ALS is initialized. Because the size and structure of the container are static, a single operation can move the all the data. The CODB holds all the exported data from all system components and routes data between components. Each component of the ALS application updates its portion of the CODB asynchronously. To further minimize the cost of the operation, we move the data between ALS components and the CODB using double-buffering.

3. REQUIREMENTS

For the ALS to provide a realistic model of airbase logistics for the DVE, it must satisfy several requirements. The first of these requirements is that the ALS provide outputs concerning sortie launch, airbase status, runway, ramp, and taxiway status, available airbase resources, personnel status, and maintenance status. The second set of requirements addresses the DIS interface. The ALS must be able to participate within a DVE that uses the DIS protocols, and send and receive DIS formatted protocol data units (PDUs). The ALS must also be able to respond to Air Tasking Orders (ATOs) that are issued by a simulated Air Operations Center (AOC) using CCSIL. ATOs are fixed-format text messages that describe targets to be struck, weapons loads, takeoff times, and other mission essential information. This information comes to the base through the wing operations center (WOC), which then passes the information on to the affected organizations on the base. To accurately portray this information flow, the ALS must be able to accept and process data that arrives from the WOC and to inform the WOC of the status of aircraft affected by current ATOs. Additionally, so that the AOC and WOC have accurate data on aircraft availability, ALS must provide information concerning aircraft availability and aircraft turnaround time to the WOC for retransmission to the AOC.

Because the ALS is a participant within a DVE, it must communicate its status to the other entities within the environment. However, the airbase’s status does not change rapidly. So, the airbase must broadcast its status at regular intervals and whenever a significant change to its status occurs. Because the airbase status is only to be reported at lengthy but regular intervals, its status only needs to be recomputed just before a broadcast occurs. If a major change occurs as the result of attack or accident, the airbase’s status should be recomputed at that time as well. The airbase’s status report must, at a minimum, contain all of the information specified within the CCSIL airbase status message so that the AOC can construct Air Tasking Orders using accurate data on aircraft, munitions, and runway availability.

Because the airbase system is intended to accurately portray airbase logistics and depict their impact on sortie generation rates, the ALS must keep track of munitions status and POL status. The system must also be able to respond to munitions resupply and POL resupply messages that arrive within DIS PDUs.

As indicated above, information generated by CWTSAR will be used to generate backtell messages to an AOC. The ALS must provide the following reports to the AOC: unit reports; unit mission status reports; takeoff, abort, and landing report; airbase status reports; airfield status reports; aircraft in distress reports; downed aircraft reports; inflight reports; and rescue unit status reports. This information will be combined with additional information in the ALS and provided to the WOC and to other participating systems.

The ALS must be able to compute the percent damage to base components and be able to simulate the destruction or damage of components such as aircraft, POL supplies, aircraft shelters, runways, or taxiways. In addition, the ALS must also adjust its sortie generation capability and
aircraft launch capability in response to damage to airbase components.

The ALS must also be able to simulate the launch and recovery of aircraft entities within the DVE. This requirement involves communicating aircraft availability status to the AOC, notifying computer-generated forces (CGF) aircraft systems that their time to launch has arrived and of the weapons load they have, and finally the ALS must respond to the CGF notification that they have returned to base and are ready for regeneration.

A final set of requirements arise from the desire to visualize the operation of the ALS. Because the CWTSAR core was designed to operate as a stand-alone constructive simulation, it does not contain a graphical component. Therefore, the ALS must interface with a system that can portray the airbase graphically. The two systems must maintain a consistent representation of the airbase’s status and must respond identically to changes in airbase status. However, only one of the systems can be used to determine damage severity and location, the other system must be adapted to accept these data as input parameters to its model.

4. ARCHITECTURE AND DESIGN

The first steps we undertook in the design process addressed the definition of the message flows between the ALS, the AOC, and the aircraft CGF generator system. These message flows are portrayed in Figure 1. The CGF system that we must interface with is the Aircraft SOAR system developed at the University of Michigan [3, 4]. The starting point for our design was to examine the required flow of data between components and then design to implement these flows between the objects.

![Figure 1: Data Flows that the Airbase Logistics System Must Support.](image-url)
AOC whenever the launch time or load required by an ATO can not be achieved. The AOC is also required to provide SOAR with data concerning required mission launch time and mission plan so that this data can be entered into the SOAR system before the ALS signals SOAR that the aircraft is ready for launch.

A challenge that we encountered with this intended message flow is that all of the systems use different techniques to keep track of aircraft. The ALS identifies aircraft by sequential number, SOAR identifies aircraft by call sign, and the AOC identifies aircraft by aircraft tail number. This problem was solved by requiring the three system to exchange data at initialization to establish the correspondence between tail number, call sign, and ALS number. While not a major problem in and of itself, this does illustrate the type of problems encountered when integrating disparate systems into a cohesive unit.

Within ALS, the integration challenges were somewhat more difficult. These problems arose largely due to the nature of the CWTSAR design. CWTSAR’s design is based upon modules and global variables. Within CWTSAR there are 23 modules, with each module having the responsibility for one or more airbase tasks. These modules are structured so that eleven main simulation tasks can be automatically accomplished by CWTSAR. These simulation tasks are facility repair, on-aircraft maintenance tasks, support equipment repairs, munitions assembly, scheduled and periodic activities, simulation of aircraft flight delay, post-flight maintenance, aircraft launch, personnel management, and resupply. These tasks occur sequentially and in a fixed order throughout the life of the simulation run.

Our ALS design was developed with the intent of modifying the underlying CWTSAR FORTRAN code as little as possible while also enabling the system to operate within a DIS-based DVE. As a result, we had to design methods for re-initializing and re-executing CWTSAR at specific time intervals, for extracting data from the system, and for eliminating CWTSAR’s capability for simulating the aircraft missions.

The design approach we took to meet these goals and the requirements listed above is centered upon the CODB architecture. By examining the required data flows between the major ALS components we were able to identify the data required by each component and the source of the data. The data is extracted from the major components of the ALS by methods in the CODB and held there until the recipient component requires it. We use the CODB to hold data between CWTSAR runs, to hold re-initialization data, and to transmit data between CWTSAR and ModSAF. Figure 2 presents the object-oriented design we developed for the ALS based upon the CODB.

Within CWTSAR, the computational models that implement the eleven major system functions were not modified because they place their results into global variables. These modules compute aircraft availability and operate to simulate sortie generation and airbase response to attack in the same manner as before they were integrated into ALS. However, to extract the results of these computations we designed CODB methods that can access the data in the global variables and place them into their appropriate containers in the CODB. These global
variables are defined within twenty FORTRAN global common blocks within CWTSAR, so in our design we use one method to extract the variables in each common block. For the purposes of initialization and re-initialization we also designed CODB methods to extract data from storage files and from incoming data from ModSAF and to insert this information into the CWTSAR global variables. In our design we use one method to extract the data from the incoming container and to place it into the global variables a single common block.

When CWTSAR finishes computing a four-hour simulation block, the data from this run is extracted from the global variables and saved to data files as well as being transferred to ModSAF for broadcast. This insures that the AOC has current data and that the data required for the next execution is available.

The data destined for transmission by ModSAF is extracted from the CODB containers and then repackaged into the correct ModSAF or CCSIL container by methods in the CODB. These methods allow CWTSAR to communicate with ModSAF while insuring that the two systems remain loosely coupled. This is important for our system because the CCSIL message formats and ModSAF utilities continue to change. The CODB ModSAF export methods are designed to place data into the format required by either CCSIL or by the ModSAF DIS broadcast utilities. Reciprocally, the data that arrives through CCSIL or via DIS PDUs is gathered from ModSAF by methods in the CODB and placed into the appropriate containers for export to CWTSAR.

A final design issue that we had to address is achieving a consistent representation of the airbase’s status between the ALS and the graphical model. The difficulties in achieving this coherence arise from the differences in representation between the two systems. The ALS does not assign damage, or even a building, to a specific location in the world. All of the buildings and facilities are placed in relative positions within the airbase but they have no coordinates within the DVE. However, the location and amount of damage are precisely specified within the model. The graphical airbase model, on the other hand, assigns facilities to a specific location within the DVE, but does not represent damage accurately. A further complication arose from the requirement that the graphical model be used to determine the location and level of damage. As a result, in our design we use CCSIL messages dispatched from the graphical model to inform ALS of an attack and the approximate location and level of damage. ALS then takes this information and by applying Monte Carlo methods assigns specific levels of damage and a location for each airbase facility that was struck.

5. CONCLUSIONS AND FUTURE WORK

With the completion of the architecture and design, we have begun implementation of the ALS system. Currently, the CODB interface to CWTSAR is complete and we are implementing the methods required to communicate with ModSAF and CCSIL. We have also started work on the messages required to insure consistency between the ALS and the airbase graphical model. The ALS system will be completely operational by March, 1997 and its first demonstration is scheduled for April.

Future work on the ALS should address two tasks, improving the interface to CWTSAR and increasing the flexibility of CWTSAR. Currently, CWTSAR is configured using data files that must be typed in. The file formats are based on the IBM 80 column punch card. As a result, establishing a configuration for an ALS execution is a lengthy and tedious process. A graphical user interface for CWTSAR configuration would simplify the process and also lead to fewer configuration errors. The other critical need is to improve the flexibility of CWTSAR. The current system is restrictive in the number of aircraft, aircraft missions, and weapon loads that it can accommodate. As a result, the ALS simulation can not accurately portray all aircraft types and mission types. This change necessitates modification to the CWTSAR code and requires re-engineering of the CWTSAR architecture as well.

REFERENCES