RAPIDLY RECONFIGURABLE HUMAN-OPERATED SYSTEMS FOR DISTRIBUTED VIRTUAL ENVIRONMENTS: REQUIREMENTS AND IMPLEMENTATION

Martin R. Stytz, Lt. Col., USAF, Ph.D.; Sheila B. Banks, Major, USAF, Ph.D. and Terry Adams, Captain, USAF
Virtual Environments Laboratory
Human-Computer Interaction Laboratory
Air Force Institute of Technology
mstytz@afit.af.mil

ABSTRACT

In this paper we identify system requirements and describe an adaptable software architecture for rapid reconfigurability. The paper discusses the reconfigurability requirements for aircraft training systems within a distributed virtual environment. We also describe the system controls, interaction support, aerodynamics model, and software architecture we designed and implemented for a Rapidly Reconfigurable Photorealistic Virtual Cockpit. Within the paper, we document the performance of the current system and suggest avenues for further research.

1. INTRODUCTION

The Virtual Cockpit (VC) is a cockpit and aircraft simulation project that addresses issues in a low-cost, portable distributed interactive simulation (DIS) capable flight simulators for aircrew training. These issues include: 1) 3D computer graphics, 2) human-computer interaction, and 3) virtual environments issues. To provide this capability, the VC requires a high resolution, visually realistic, functional portrayal of the cockpit interior and a user interface for immersive operation of its controls. The user interface must support a range of interaction that permits useful work to be accomplished in a manner that emulates the operation of the actual aircraft’s controls. The VC has only minimal physical instrumentalities for control of the aircraft, typically only a throttle and stick. All other components of the aircraft cockpit have only a 3D graphical representation. In an earlier paper by McCarthy, et. al. [1], we reported on the initial architecture of the VC and our implementation of its sensors, DIS capability, and flight dynamics. Here we report on our progress in developing a rapidly reconfigurable VC that can rapidly switch, or reconfigure, between different types of aircraft.

Reconfigurable training systems have been used for many years to reduce the costs associated with acquiring and maintaining a broad-based training capability. In these systems, reconfigurability is achieved by swapping physical components of the simulator, thereby changing the controls and displays to match those of the system to be simulated. We believe that the technology will soon be available to enable the emulation of displays and controls using 3D computer graphics models. We believe that the availability of rapidly reconfigurable systems for distributed virtual environments (DVEs) may reduce the costs associated with inserting human-operated entities into the DVE.

To develop a system to support rapid reconfigurability between virtual systems, a number of technical issues must be solved. These issues include presentation of the controls for the system, methods for interaction with the system’s controls, achieving physical motion and sensor fidelity for the systems, defining a system software architecture that can support rapid reconfigurability, and achieving correct weapon system performance. Because the modeled systems exist in the real world and the DVE application is used for training, these issues must be solved in a way that preserves the training value and fidelity of the modeled systems. We addressed these issues in the context of developing a prototype Rapidly Reconfigurable Photorealistic Virtual Cockpit (VC).

The VC is a virtual aircraft simulator prototype intended for use by military pilots for tactical training at their individual units. The VC can model different aircraft and insert any of them into a Distributed Interactive Simulation (DIS) based distributed virtual environment. To accurately model the aircraft for both the user and other DVE participants, the VC must present realistic cockpit displays and utilize accurate aerodynamics, onboard sensor, and weapons models. Because the only hardware devices available within the VC are the stick and throttle, the remainder of the cockpit interface is activated using mouse inputs. While using a mouse decreases the realism of the VC interface, it is necessary given current technology. To increase the versatility and usefulness of the VC it is implemented on commercially-available hardware.
The next section presents relevant background for the results reported in this paper. Section Three summarizes the design of the reconfigurable VC. Section Four discussed our implementation. Section Five contains results and recommendations for future work.

2. BACKGROUND

Previous work on the Virtual Cockpit [1, 2, 3, 4] addressed two issues: 1) increasing the realism of the appearance of the interior of the VC and 2) improving the user’s interface to the virtual environment. The first issue is important because it increases the presence of the pilot users by placing them within familiar surroundings and places the controls in their accustomed positions. The second problem addresses the need to provide a functional interface to the system’s controls in the VC using available technology. The first problem entailed developing a photo-realistic virtual cockpit with working dials, switches, and circuit breakers. The second problem required developing techniques to interact with the VC using its photo-realistic controls.

Three factors that can impart a realistic look to the VC are the visual fidelity of the objects depicted, the accuracy of their physical motion, and the frame rate. We define fidelity as the amount of detail represented in the object. Fidelity increases as the amount of detail increases up to the limits of the human visual system. In our VC, photorealistic portrayal requires the most detailed models the hardware can support. If the VC had no other requirements, a highly realistic model could use tens-of-thousands of polygons to portray the forward crew station. However, portrayal of model detail directly competes with computational resources for: 1) simulating the physics of flight, 2) collision resolution for interaction with other entities in the virtual environment, 3) portrayal of complexity of the environment (outside the cockpit), 4) support for human-computer interaction, 5) portrayal of other virtual environment actors, and 6) network traffic requirements. To achieve acceptable visual realism and motion in the depiction of the VC’s cockpit interior we used a combination of polygonally-defined objects and texture mapping. The polygonal objects form the basis of the VC’s instruments, consoles, switches, and knobs.

The VC’s architecture is based upon the Common Object DataBase (CODB) as described by Stytz [5]. The Common Object DataBase is primarily 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 must only access the container in the CODB where the information resides.

In the VC, the CODB provides an inheritable, reusable interface to the computer hardware, input and output devices, and Silicon Graphics’ Performer software. When the VC begins execution, the CODB loads the terrain description for the distributed virtual environment and 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. Finally, the CODB provides an interface to reusable and inheritable classes for a variety of input devices. The CODB does not compute motion, pitch, or heading; these computations are the responsibility of the VC and its aerodynamics unit.

The VC’s DIS interface is our Lab’s World State Manager 3.0 (WSM). The WSM provides both a protocol data unit send and receive capability. The World State Manager provides containers for broadcasting aircraft and weapons entity state PDUs and for receiving entity state PDUs.

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 VC is initialized. Because the size and structure of the container are static, a single operation can move the data. The CODB holds all the exported data from all system components and routes data between components. Each component of the VC application updates its portion of the CODB asynchronously. To further minimize the cost of the operation, we move the data using double-buffering.

The Virtual Cockpit operates within distributed virtual environments [6]. A distributed virtual environment (DVE) is a large-scale, networked, computer-based, virtual world wherein several thousand human and computer controlled entities can interact. Entities move within the virtual environment using aerodynamics, celestial mechanics, vehicle dynamics, and soil mechanics models to compute their motion. The environment requires models of the terrain and other features such as roads, forests, rivers, bridges and buildings. The technologies supporting distributed virtual environments are continuously challenged by requirements for increasing complexity (in both numbers and types of entities), for increased fidelity, for improved situation assessment and user interaction, and by changing computer platforms.
In the next section, we present the requirements for the reconfigurable VC and discuss the design we developed.

3. RECONFIGURABLE VIRTUAL COCKPIT REQUIREMENTS AND DESIGN

As the starting point for our research, we began with the existing Photorealistic VC described by Stytz and Diaz [4]. This application, while suitable for supporting photorealistic portrayal of aircraft cockpit interiors, could not support rapid reconfigurability. As a result, we decided to re-implement the VC using the CODB as the core of the architecture and design. To aid us in the development on the new VC, we developed a set of requirements for the software architecture, cockpit geometry models, system components, and user interface. These requirements are presented in Table 1. The design that resulted from these requirements is presented in Figure 1. The remainder of this section describes how we designed the reconfigurable VC to meet these requirements.

### Table 1. Rapidly Reconfigurable Virtual Cockpit Requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reconfigurable Software Architecture</td>
<td>Allow rapid reconfiguration of both aircraft geometry, aerodynamics, sensors, and other simulation components</td>
</tr>
<tr>
<td>1.2</td>
<td>Support switching within one frame update. Maintain a minimum 15 Hz update rate</td>
</tr>
<tr>
<td>1.3</td>
<td>Provide for notification of the entity type change to other participating systems in the DVE</td>
</tr>
<tr>
<td>1.4</td>
<td>Support multiple aircraft</td>
</tr>
<tr>
<td>2. Reconfigurable Cockpit Geometry Models</td>
<td>Allow switching between different aircraft cockpits</td>
</tr>
<tr>
<td>2.2</td>
<td>Reuse existing F-15 cockpit model geometry</td>
</tr>
<tr>
<td>2.3</td>
<td>Create photo-realistic F-16 Instrument Panel</td>
</tr>
<tr>
<td>3. Reconfigurable Simulation Components</td>
<td>Develop reconfigurable aircraft aerodynamic model</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Radar must be able to change field-of-views while running</td>
</tr>
<tr>
<td>3.3</td>
<td>Develop a reconfigurable weapons controller</td>
</tr>
<tr>
<td>4. Improved Cockpit Interface</td>
<td>Allow selection of three dimensional panels and instruments</td>
</tr>
<tr>
<td>4.2</td>
<td>Eliminate need for maintaining an active panel for button selection</td>
</tr>
<tr>
<td>4.3</td>
<td>Improved switches and dials</td>
</tr>
</tbody>
</table>

The design we developed uses the CODB as its core, all data that moves between components passes through the CODB in a container. The Airplane class encapsulates all of the aircraft functionality. Within the Airplane class are the classes for the aircraft aerodynamics (CODBAeroModel), the aircraft weapons controller, the Radar, and the F-15 and the F-16 instrument panels. The VC Renderer class manages all of the rendering functions, and relies upon Performer to execute most of the culling and drawing operations as well as to maintain the scene database. The Input/Output (I/O) classes provide a common interface to the devices in the lab. Finally, the World State Manager provides an interface to the DVE and to the data from the other entities.

![Figure 1: Virtual Cockpit Software Design.](image)

(Diamonds represent aggregation, triangles represent inheritance. Numbers in circles beside the components represent the primary CODB data structure updated by this component.)

Our main criteria for the software architecture are that it must be able to support the development of multiple aircraft cockpits and allow reconfiguration between them. The architecture must also be able to quickly switch between aircraft configurations with minimum impact to performance and be able to support the wide range of functionality of different aircraft. We can satisfy these requirements using the CODB architecture.

The container abstraction within the CODB greatly assists in achieving reconfigurability for the VC. For instance, if the VC is initialized to switch between a C-5 and a F-15, it will probably require models of each aircraft’s navigational system. As a result, two separate software models must communicate with the rest of the VC and share state information. By communicating with the radar models through a common container we simplify the process of moving data to and from the navigational system model.

The cockpit geometry modeling task consumed a great deal of time. Crafting accurate geometric models of a cockpit configuration is a challenge even with the most modern drawing tools that are available. Because of the time involved, we decided to reuse the geometric model for the previous F-15 VC and implement only one other type of aircraft, an F-16. We use two separate instrument panels to
represent the F-15 and F-16 cockpits because there are significant differences in appearance and functionality between the two cockpits. In our design, the current instrument panel depends upon the current aircraft type. The most technically challenging aspect of this process was texture creation and modeling.

The textures were created using a white text color that when placed on a colored polygon allows the text to take on the color of the texture-mapped polygon. The non-text portion of the texture is totally transparent; allowing it to take on the color of the background polygon (the polygon behind the texture-mapped one). By using the correct coloring scheme and the alpha component of textures, these same text-based textures can be used in many different cockpit models.

The reconfigurable models required for this project are an aerodynamics model, a weapons controller, and a radar. The performance and accuracy of these components is critical to achieving a reconfigurable system that has an acceptable level of fidelity. Of these three systems, the aerodynamics model is the most crucial to achieving fidelity because its performance is immediately noticeable to the user. Reconfigurable software models provide a single point solution for multiple modeling situations. Secondary reconfigurable models included a simple radar model and the modification of the weapons controller to allow multiple aircraft to use it independently.

We were able to achieve our fidelity goal by using an aircraft aerodynamics model developed by Wright Laboratory. This model is parameterized, allowing us to use data files to determine the type of aircraft it will represent in the DVE. The original model initializes as a single type of aircraft and is flown as that aircraft for an entire simulation. To use the model to rapidly reconfigure from one type of aircraft to another the model underwent several transformations. We changed the model to operate with the VC’s stick, throttle, and rudder input devices. We also changed the aerodynamics model to allow the model to initialize the position of the aircraft to any orientation or position and we changed the input and output modes for the model to interface with the CODB architecture. After making these changes, multiple aircraft types can be created using this single model.

A simple radar model design provides the basic radar functionality that we require. The radar class design is depicted in Figure 2. The design encapsulates the radar in its own class and provides methods for setting radar field of view and range.

4. RECONFIGURABLE VIRTUAL COCKPIT IMPLEMENTATION

The first step in moving the 1995 VC into the CODB architecture was to break apart the VC into its major components. These components were identified as the cockpit, weapons, head-up display (HUD), aircraft model, and multi-function display (MFD) / inertial navigation system (INS) / radar. The design of the 1995 VC tightly coupled these classes and they communicated via a shared memory structure. Once the 1995 VC was broken down into its components, the cockpit component was chosen as the first component to integrate into the VC. The remainder of the components were readily integrated into the CODB architecture.

The second group of components integrated into the VC were the HUD component, the Multifunction Display (MFD), the Inertial Navigation System (INS), and the Radar components of the F-15 VC. The HUD was changed to read the aircraft state from the CODB. The MFD and INS were also integrated into the CODB architecture so that their input and output data came from the CODB containers. The process was straightforward, in that all shared memory accesses were replaced with CODB accesses.

After all components were moved to the CODB architecture an Airplane class was created to provide a framework for both the individual aircraft components and the reconfigurable components. The Airplane class is a multi-processing application component and is created with an Update method and a Draw method for Performer. The components are split between F-15 components and F-16 components. For reconfigurable components, such as the aerodynamics model, only a single type of class reference is needed and is called regardless of the type of aircraft (F-15 or F-16) being flown. For non-reconfigurable classes, such as the instrument panel, we use a case statement to call only the class associated with the current type of aircraft being portrayed.
A second reconfigurable model was created to allow customization of the radar for several different aircraft. The new radar class uses the container that holds all DIS entities, in flat-earth Performer coordinates, from the CODB as input and places its output in the Radar’s output container. Within the Radar class, all active aircraft tracks are evaluated for their relative azimuth, elevation, and range to the VC. If a track is within the radar’s field of view then that track is activated, otherwise the track is left in the inactive state. The model is reconfigured by using the methods within the Radar model to match the operation of the Radar class with the specific aircraft’s field of view.

Implementation of a reconfigurable weapons controller was much simpler than for the aircraft or the radar models. The weapons controller class provides an interface to cannon, bomb, and missile classes for the simulation. The weapons class uses a file to initialize the number of weapons and their position in the aircraft. These changes were straightforward.

Replacement of the VC’s user interface focused on simplifying the Pod and on supporting natural interaction with the cockpit’s virtual environment. The Pod consists of several classes that must be inherited and components that must be over-ridden, resulting in a complicated class that is difficult to use. To expand the interaction capability we developed a class based on Performer’s geometry picking functions. The class provides a way of determining if a simulation object was selected by the user or if a portion of a larger object was selected. The class is able to accurately pick irregularly shaped objects instead of being limited to picking against a square area.

The additional capabilities provided by the Selection Manager for the F-16 cockpit provide interaction techniques that are more dynamic then those in the F-15 cockpit. The new selection manager allows a user to select cockpit switches or dials with multiple mouse buttons. This capability provides the developer with the ability to move switches counter-clockwise if the left mouse button is pressed and clockwise if the right mouse button is pressed.

Dials were implemented using a GL vector font; numbers on the dials were updated based on the instrument’s current setting. However, the dials did not spin, which prevented the pilot from noticing the motion or the rate at which the value was changing unless they were staring directly at the values as they changed. Therefore, spinning dials were implemented using Coryphaeus Software’s Designers WorkBench (DWB) to provide a realism not easily duplicated using the GL calls. Finally, the cockpit utilizes many DWB models to increase the realism of the F-16 VC.

Receiving entity state information is the most complex portion of the DIS interface because the entities must be converted into the local simulation’s world coordinate system and then displayed accurately to the pilot. This follows a four step process: 1) Read the incoming entity container in the CODB, 2) convert the DIS coordinates into Performer flat-earth coordinates, 3) store the coordinates in local entity container, and 4) update position and orientation of the aircraft in the Performer tree.

The WSM allows us to send entity state information with minimal programmer effort. The Airplane class broadcasts its positioning by sending its entity state information to the WSM. In the WSM, the coordinates are converted from Performer flat-earth to DIS coordinates for transmission. The output container is also updated to reflect the current aircraft being modeled by the VC. The output container is updated every frame update. The WSM uses dead-reckoning to determine when to send out a packet on the network.

The final step of DIS integration was sending out entity state information for VC weapons. The same process is followed by the weapons for sending out entity state information; however, DIS broadcast only happens after the weapon has been dropped or fired.

5. RESULTS AND FUTURE WORK

The VC’s ability to rapidly reconfigure allows a single application to represent several different aircraft depending on unit or individual training requirements. The VC currently models F-15 and F-16 aircraft, including their cockpit instrument panels and weapons, aerodynamics, and sensors on a single workstation. The VC is based upon an object-oriented design, our lab’s object-oriented run-time data repository (the Common Object DataBase), a loosely coupled system architecture, and a highly-parameterized aircraft aerodynamics model. Aircraft aerodynamics model parameterization allows for the seamless addition of new aircraft models into the VC.

The reconfigurable computer architecture performs as intended. We achieve 15 or more frames rendered per second and the switch between aircraft types takes less than a single frame update. The selection of the CODB as the framework for simulation components greatly simplified the components’ interfaces, the system integration effort, and testing.

Figures 3 and 4 present the cockpit interior for the VC when it is in F-15 mode. Figure 3 presents a view of the complete cockpit interior. Figure 4 presents a closeup of the front left panel showing the level of detail we achieved.
Figures 3 and 4 present a view of the Virtual Cockpit interior when it is in F-15 mode.

Figures 5 and 6 present a view of the Virtual Cockpit interior when it is in F-16 mode.

Reconfigurable simulation components greatly eased the development of the cockpit. The models are more difficult to develop than non-reconfigurable models but this difficulty pays dividends in reduced overall development time. For the VC the fidelity provided by the reconfigurable aircraft model was more than acceptable. The accuracy and performance of the reconfigurable aircraft model is very good, several pilots have commented on its fidelity. The radar model developed for the F-16 VC provided basic functionality and is another component that can be made more realistic and reconfigurable. A radar model could be made reconfigurable in the same way as the aircraft model, by allowing all its performance, not just the field of view, to be determined based on aircraft and radar type. Reconfigurable models give the developer a step up in developing a new aircraft simulation; however, they are not always the best choice. The instrument panels in aircraft are different enough that they should not be reconfigurable but instead should be modeled individually.

There are several areas of reconfigurability that remain to be investigated. The development of separate VC aircraft could be greatly increased by utilizing more reconfigurable simulation components. Additional common aircraft components (infrared, counter-measures, etc.) should be identified and reconfigurable models developed to emulate their functionality.

The VC should also be expanded to allow a single aircraft to represent multiple aircraft entities in an exercise at the same time. The VC now switches between aircraft, so that it only inserts one aircraft into the DVE at a time. However, by allowing a single pilot to control multiple aircraft, much larger mission scenarios could take place with fewer pilots. Of course, some form of computer-controlled actor capability is needed to fly each of the aircraft when they are not under human control, such as that described by Edwards [7] or Tambe [8].

REFERENCES


