REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(LEAVE BLANK)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	3 August 1999	Professional Paper

4. TITLE AND SUBTITLE

The ACETEF HLA Interface for JADS-EW

5. FUNDING NUMBERS

6. AUTHOR(S)

Lawrence Conrad

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Air Warfare Center Aircraft Division
22347 Cedar Point Road, Unit #6
Patuxent River, Maryland 20670-1161

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Air Systems Command
47123 Buse Road, Unit IPT
Patuxent River, Maryland 20670-1547

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper presents the software approach taken at ACETEF to support the JADS-EW test. It begins by describing the overall structure of the JADS-EW federation and the roles played by the federates. The HLA interface consisted of two major components, namely the RTI Interface and the SWET Interface, designed to work together but separately, in order to decrease the workload of a single HLA interface. Though performing different functions, their general structure is similar in that each is guided by a Federate Manager, which directs all activity according to the specifications of a particular federation. Another common feature between both interfaces is the utilization of the C++ class inheritance, virtual functions, and polymorphism capabilities, which greatly assist in producing highly maintainable and reusable code.

14. SUBJECT TERMS

Joint Advanced Distributed Simulation Electronic Warfare
HLA Reusability Maintainability Interface Design C++

15. NUMBER OF PAGES

7

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

The ACETEF HLA Interface for JADS-EW

Lawrence A. Conrad
Naval Air Systems Command, Atlantic Ranges and Facilities
Air Combat Environment Test and Evaluation Facility
Patuxent River, MD 20670
(301) 342-6365
conradla@navair.navy.mil

Keywords:
HLA, Reusability, Maintainability, Interface, Design, C++

ABSTRACT: *This paper presents the software approach taken at ACETEF to support the JADS-EW test. It begins by describing the overall structure of the JADS-EW federation and the roles played by the federates. The HLA interface consisted of two major components, namely the RTI Interface and the SWEG Interface, designed to work together but separately, in order to decrease the workload of a single HLA interface. Though performing different functions, their general structure is similar in that each is guided by a Federate Manager, which directs all activity according to the specifications of a particular federation. Another common feature between both interfaces is the utilization of the C++ class inheritance, virtual functions, and polymorphism capabilities, which greatly assist in producing highly maintainable and reusable code.*

1. Introduction

The Air Combat Environment Test and Evaluation Facility (ACETEF) located at the Naval Air Warfare Center Aircraft Division (NAWCAD), Patuxent River MD, recently contributed to the Joint Advanced Distributed Simulation Electronic Warfare (JADS-EW) test. ACETEF is organizationally part of Atlantic Ranges and Facilities and directly supports Naval Aviation Systems Command. For ACETEF, this exercise necessitated the replacement of the HLA interface from the Engineering Prototype Federation by a new HLA interface. This interface was required, of course, to interact with the RTI as well as interact successfully with various legacy code. Although this was a JADS exercise it is not the intent of this paper to report officially on the JADS-EW test or describe it in detail, but will instead mainly be limited to the role of ACETEF with respect to JADS-EW and the software interface employed.

2. The JADS-EW Federation

2.1 Background

The purpose of the JADS-EW test was to yield data which would determine the usefulness of HLA for long distance distributed simulations of a closed-loop nature. The comparison of data obtained through a live exercise with that obtained through an HLA federation would shed much light on the matter. In order to accomplish this without undo complexity, the JADS-EW test scenario had to be brief and simple to repeat, and easily translated into HLA terms. The test chosen, then, was to measure the performance of a Radio Frequency (RF) jammer, aboard an F-16 aircraft, as it was tracked and (virtually) fired upon by a series of hostile landsites. The jammer must sense the RF from the threat landsites, and produce its own jamming RF in response. This RF would, in turn, be sensed by the landsites, affecting their tracking and missile performance (as well as the F-16's survivability). Such a scenario could easily and quickly be repeated, both during the live exercise and via simulation over the HLA, producing a large amount of data for analysis. Although the jammer's performance and the F-16's survivability were critical measures of performance, it must be remembered that the jammer was not the actual subject of the test. The purpose again, was to determine whether data obtained through live players on a range could be statistically replicated by a combination of man-in-the-loop, hardware-in-the-loop, and software devices, over an HLA federation.

2.2 JADS-EW Phases

The JADS EW test had three distinct phases.

Phase 1) The collection of real time data of an F-16 flying past four surface-to-air threats on a test range. The threats emitted RF and the jammer aboard the F-16 reacted accordingly. Telemetry data was captured as well as the jammer and threat events, and other RF environmental characteristics. This created a baseline set of data for comparison with the HLA-derived data.

Phase 2) A jammer simulator was located at ACETEF. Through the RTI, threat mode codes and F-16 positional data was used to digitally stimulate the jammer simulator. Jammer activity from the simulator was sent back to the threats, to support a closed-loop simulation. ACETEF involvement was minimal.

Phase 3) The JADS federates from phase 2 remained virtually unchanged but now ACETEF was fully involved. The jammer simulator was removed. ACETEF used the threat and aircraft position data received via RTI to drive actual RF emitters in an anechoic chamber, which in turn, would stimulate the jammer located onboard an F-16 hanging in the chamber. The Simulated Warfare Environment Generator (SWEG) represented the landsites, aircraft and emitters in SWEG-space and served to control the RF emitters. SWEG had to continually adjust the power of each emitter according to the distance and tracking error of each emitter to the F-16. Jammer activity was again sent back to the other federates to establish the closed-loop simulation. In addition, the stimulator status (power levels, RF states) was sent to the other federates to assure them that the RF stimulators were reacting properly to the virtual geometry of the threat in relation to the aircraft.

2.3 JADS-EW Members

The organizations contributing to the JADS-EW test were:

1) JADS Test Control Facility (JADS TCF). Central data collection point and Federation controller. Located at JADS office in Albuquerque, New Mexico.
2) Air Force Electronic Warfare Environment Simulator (AFEWES) in Fort Worth, Texas. Simulated threats.
3) Georgia Technical Research Institute (GTRI). Provided technical assistance, the jammer simulator (Phase II only), data analysis products, and the digital interface to the jammer (hereafter referred to as the Digibus Monitor).
4) Air National Guard, Air Force Flight Test Center, Tucson, AZ. Provided the F-16.
5) ACETEF
6) 413th Test Squadron, Edwards AFB, CA, provided the range support.

2.4 JADS-EW Federates

The federates consisted of:

1) Platform. This generated the F-16 flight path. Located at JADS.
2) RF Environment. Replicated the extraneous RF signals that were present on the range during phase 1. Located at JADS.
3) Threats. Generated mode codes corresponding to the RF signals from the threats. Also produced the missile fly-outs. Located at AFEWES.
4) Terminal Threat Handoff. Controlled the threats' engagements. Located at JADS.
5) Test Control Facility, located at JADS.
6) Jammer. The ACETEF federate.

3. HLA Interface Architecture at ACETEF

3.1 Functional Requirements

The HLA interface was designed to support not only the requirements posed by JADS-EW but to have the ability to support any future exercise. Such a design requirement is greatly satisfied by the C++ class structures, as will be shown.

The following are the JADS requirements for the ACETEF HLA Interface:

- Respond to TCF orders to stop/start test
- Accept missile, aircraft entity states and pass on to SWEG
- Accept threat mode code changes and pass on to SWEG and RF Stimulators
- Accept Jammer monitor data and pass on to RTI
- Accept Chamber monitor data and pass on to RTI
- Periodically build a link-health message and pass on to RTI
- Accept link health messages from other federates and display
- Flag missing or out of sequence data from RTI and react accordingly
- Dead reckon aircraft in case of delayed or missing position updates

3.2 Interface Architecture

In order to decrease workload on the HLA interface ACETEF decided to separate this interface into two major processes (the RTI Interface and the SWEG Interface). Thus, the RTI Interface and the SWEG Interface were designed to cooperatively fulfill the functionality of the HLA Interface requirements. Each would periodically scan a section of shared memory for instructions issued by the other, all the while being subject to RTI and SWEG callbacks. For the purpose of JADS, no intensive callbacks from SWEG were expected—mainly the acknowledgement of a new entity into, or a deletion of an entity from, the scenario—but in future projects, it is entirely certain that SWEG will be issuing large amounts of callbacks (positional updates, births, deaths, weapons firing, etc.).

3.3 JADS Data of interest to ACETEF

According to the fed file, (a common file used by all federates which lists in detail for the RTI the types of data it is expected to transmit) and the JADS EW Federation Object Model (FOM), there were four RTI object classes and four interactions being communicated, to and from ACETEF.

3.3.1 RTI Objects used

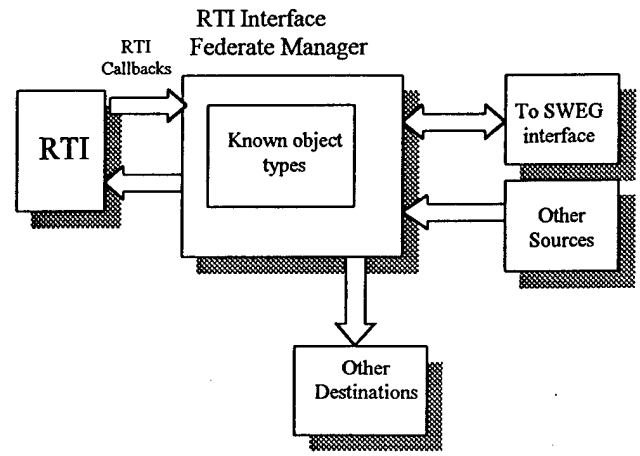
- Class Node, with attributes:
  - Execution Control – the start and stop commands from TCF.
  - Link Health Check – two words describing a Node's status and its view of the federation's status
- Class Aircraft, with attributes:
  - Live Entity State – similar to DIS standard identity description, and positional, velocity, acceleration, orientation information
  - Button Discretes – one word indicating switchology in the cockpit (whether the jammer is turned on or off by the pilot)
  - Air Frame Configuration – one word indicating a particular configuration the aircraft was in. Not used by ACETEF.
  - Threat Performance – this was unusual in that it was actually the Threat Performance data from one of the landsites, but it was to be published by the aircraft.
- Class Landsite, with attributes:
  - Entity State—virtually the same as the aircraft Live Entity State. Not used by ACETEF. Locations of landsites are known at startup.
  - Threat Performance – This structure gives the error between the aircraft's actual location and the position that the landsite believes the aircraft has.
- Class Missile, with attributes:
  - Entity State
  - Missile Performance – structure yielding miss distance after detonation. Not used by ACETEF

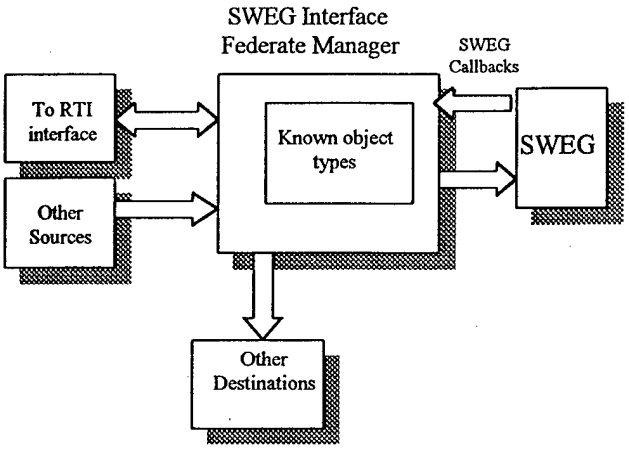
3.3.2 RTI Interactions used

- MultiSpectral Source Mode Changes class, with parameter:
  - Message – gives the landsite source ID plus the a code corresponding to an emitter's mode change. Received at ACETEF.
- SUT Jammer Technique Command, with parameter:
  - Message – gives the jammer response to RF. Generated by ACETEF.
- SUT Receiver Track Update, with parameter:
  - Message—gives the jammer's view of the RF world. Generated by ACETEF.
- User Defined QA Data, with parameter:
  - Message—gives the chamber status. Generated by ACETEF.

3.4 Basic Structure

The RTI interface and the SWEG interface are similar in general structure. Each has a customized *Federate Manager* class for the current project, derived from a base Federate Manager class, which guides the overall behavior of the processes.Each have C++ object classes and C++ interaction classes which somewhat correspond to the RTI Object and Interaction classes as described in the fed file. These C++ classes have far more in them, however, than their corresponding fed file classes. Not only do they contain structures which replicate the data being passed, but they naturally have C++ methods to handle the data and many other variables used for internal bookkeeping. For any particular federation, these C++ classes will likely be derived from base classes to handle any specialized needs. For instance, in the ACETEF interface we have a base class named "Object" and a derived class named "Entity State Object", but for JADS-EW new classes had to be created to handle the unique needs and were named, "Jads_Aircraft", "Jads_Landsite", etc. (For examples of the JADS-EW unique needs, see section 3.5.) The following figures show how the Federate Managers interact with other components of ACETEF and the RTI.





Although the figures above are very simple, they demonstrate that control is centered in the Federate Managers of the RTI and SWEG interfaces. The Federate Manager will know which derived object/interaction class to use for every object/interaction encountered. It ensures that the appropriate method is called to handle the data.

The base classes for all federations, then are:

- Federate_Manager class
  - Controls/accepts startup/shutdown commands
  - Initialize shared memory, link to SWEG, link to other ACETEF systems
  - Publishes and subscribes according to the publish and subscribe methods of the derived classes
  - Handles Federate Ambassador callbacks (RTI Interface)
  - Handles SWEG callbacks (SWEG Interface)
  - Can have any number of completely unique functions
- Object/Interaction Classes
  - Selected to correspond to the fed file and FOM
  - Publish and Subscribe methods
  - Pass data to/from RTI or SWEG
  - Pass data to/from other ACETEF players
  - Can have any number of completely unique functions
- Tracker Classes
  - Watch for missing data or timeouts
  - Easily customizable

3.5 Derived Classes used for JADS-EW

For the RTI and SWEG Interfaces, the derived classes were based similar hierarchies, although the particulars were different.

3.5.1 Customized Behavior for JADS-EW

- Federate Manager:
  - Use a Jads_Node Object
  - Publish link health check once a second
  - Use tracker objects to check for missing data or failure of another system within ACETEF
  - Dead reckoning for Aircraft if time between updates exceeds a maximum value
  - Communicate via TCP/IP with the Digibus Monitor
  - Know all derived classes
  - Pass data to SWEG
  - Pass data to other ACETEF systems
- Jads_Aircraft
  - Pass entity state to SWEG
  - Pass button discrete message to jammer controller
  - Handle (ignore) airframe configuration attribute
  - Handle incoming Aircraft Threat Performance attribute. Use it to represent the tracking error of one particular landsite.
- Jads_Landsite
  - Use Threat Performance data to drive a "ghost" aircraft in SWEG. (Because each landsite has its own belief of the target's position, the RF power actually received by the aircraft from each landsite will be affected by the error in position. In order for SWEG to attenuate the RF emitters accordingly, SWEG must have additional aircraft in SWEG-space whose positions

correspond to the tracking error of each landsite.)

- Jads_Missile
  - Pass entity state on to SWEG

- Other: Interactions

Each interaction used for this exercise was customized for JADS-EW. These included the MultiSpectral Source Mode Changes Interaction, which was used to turn on and off the various emitters in the chamber, the communication from the digibus monitor and that from the chamber monitor.

3.5.2 Use of Derived Classes. Examples.

Derived classes are more than just an organizational arrangement of objects for no particular benefit. They have a great power behind them, and that is evident through virtual functions and polymorphism. If the classes are constructed carefully, it greatly simplifies the "generic" portion of the code. That was the goal in writing the ACETEF interface for the JADS-EW test. This is perhaps best described by some simple examples.

3.5.2.1 Example 1. The Main Program

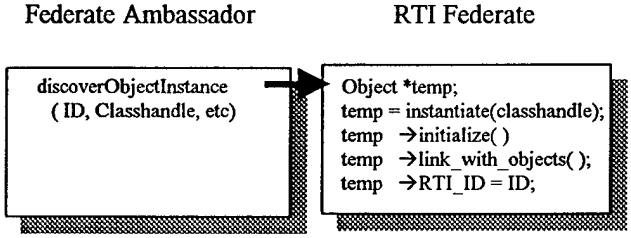
The following C++ code is a general representation of the initialization procedures and the main loop within the RTI Interface, but is neither complete nor exact. However, it demonstrates how the Federate_Manager is in control of all activity.

```
#include [all the usuals]

// BASE CLASSES
#include "Federate_Manager.h"     // Manager Class
#include "Objects.h"              // Objects Class
#include "Entity_State_Objects.h" // Derived from
                                  // Objects
#include "Interactions.h"
...

// DERIVED CLASSES
#include "Jads_Federate_Manager.h"
        // defines Jads_Federate_Manager, a derived
        // class of Federate_Manager"
#include "Jads_Objects.h"
        //which includes each object used, like
        // Jads_Aircraft.h
#include "Jads_Interactions.h"
        // which includes Jads_mssm.h, etc

void main(void) {
Federate_Manager *fed; // use pointer to the base class
                       // Federate_Manager

fed = new Jads_Federate_Manager ;
        // as described in Jads_Federate_Manager.h

fed→ initialize();
        // Sets up SWEG, or the chamber, or anything
        // required.
fed→ join_federation();
fed→ publish_and_subscribe();
        // calls each P & S method as described by
        // each derived class being used. Uses
        // RTI methods to determine constants like
        // "Jads_Federate_Manger::aircrafthandle"
fed→ wait_for_start();
        // could wait for outside start command here,
        // or just be blank. Or anything.

while(1)
        {
        tick(.001, .002);
        scan_shared_memory();
                // see if SWEG interface has anything
                // to say.
        if (Federate_Manager::quit) break;
                // "quit" a static member variable for
                // class. Could be set various ways.
        fed→specialA();
                // use to check Digibus Monitor,
                // update Link Health Check to RTI,
                // check for data dropouts of aircraft,
                // and initiate dead reckoning if needed,
                // or do anything, or nothing!
                // The Parent class, Federate_Manager,
                // has this as a blank method.

        } // while
....
} // main
```

What does this buy you? One might say that it buys you little since you still have to do all that coding *someplace*. Yes, a lot of coding goes into the Federate_Manager class and also the Jads_Federate_Manager class. But it buys you organization, reusability and readability. For the next federation, in this code segment, change only a few different includes to use, and the "fed = new [some type of Federate_Manager]" line. The rest is built block by block in the classes, and are based upon parent classes with examples, a recommended structure, and method names and uses readily apparent.

3.5.2.2 Example 2. discoverObjectInstance( )

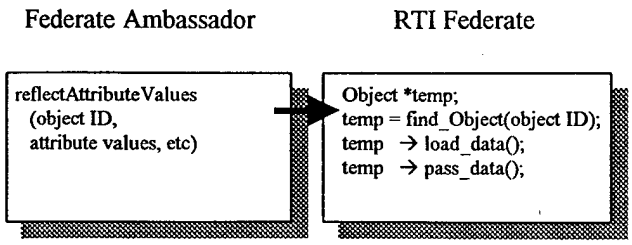
In this example, the Federate Ambassador returns a callback of discoverObjectInstance(). This callback provides to the RTI interface the object ID handle and the class handle. The Federate Manager will then instantiate a C++ object according to the programmer's instructions. For instance, in JADS-EW, the RTI Interface instantiates a Jads_Aircraft when the discoverObjectInstance() arrives with the class handle being an aircrafthandle. The exact value of aircrafthandle is determined at run time and needn't be known in advance. Control is then passed to the Jads_Aircraft constructor and initialization methods, which reserve location in shared memory for future data updates.



Inside Jads_Federate_Manager::instantiate(), a Jads _Aircraft is instantiated when a handle of value aircrafthandle is inputted. The pointer to the newly created Jads_Aircraft is returned and temp takes its value. Now, the initialize() method called will be the method created especially for a Jads_Aircraft object. The newly created object, a Jads_Aircraft, is put into the list of instantiated objects and the RTI Object ID is stored within it. (Actually, the link_with_objects() and the setting of the RTI_ID occur in the initialization routine, but for the purpose of illustration, I have included them here.)

3.5.2.3 Example 3. reflectAttributeValues()

Here, the Federate Ambassador provides to the RTI Interface the object ID and a block of data representing its attributes. The Federate Manager must examine its lists of instantiated objects and return a pointer to the correct one. From that moment on, the Federate Manager has absolutely no interest in what thing is being updated, it is all handled by the pointer.



The beauty of this method, like the example before it, is that the Federate Manager need never know what "temp" represents. The Federate Manager merely finds which object is being updated (from the ID value), returns a pointer to it (whatever it is), and the rest happens within the object's methods. This will work as long as the pointer "temp" is a pointer to an object of the base class and that all derived classes contain methods with standard names. What happens to be in those methods is completely customizable by class and is irrelevant to the Federate Manager. This yields code which is extremely reusable, in that the only things that change much are the outlying classes, not the main program nor it's organization.

4. Visual Features Employed

4.1 SWEG

The SWEG engine provides an excellent two-dimensional representation of all the players in SWEG-space during Phase 3. Each landsite was drawn in a different color and each missile shot from the landsite and the ghost aircraft associated with it used the same color. When a landsite truly had no idea where the aircraft was, it was obvious because the ghost aircraft for that landsite was nowhere near the actual target.

In addition to the SWEG graphics, we employed two other useful graphical tools.

4.2 The Link Health Check Monitor.

Each bit of the link health message had a meaning. Each was identified with a particular device, or communications link. For instance, at ACETEF, one bit indicated the status of SWEG; another, the status of the link to Fort Worth; another, the status of the digibus monitor, etc.

The RTI Interface shared this status information with a another running process through one word in shared memory. This program read the word and graphically displayed the contents on screen.

4.3 The Emitter Monitor

The SWEG Interface also shared a block of shared memory with another process. This was the Emitter Monitor and it read a word which reflected the on – off

state of each of the approximately 30 RF emitters being driven by AFEWES.

The Link Heath Check Monitor and the Emitter Monitor had negligible overhead and greatly assisted in debugging and situational awareness.

5. Latency Performance

The ACETEF HLA Interface performed well. It was a complex system, divided into two major processes, which had to communicate to many pieces of legacy and new code, and timing was critical. Because of the closed-loop nature of the test, the latency specification was critical. Testing showed that less than 30 milliseconds were needed to effect a RF emitter mode change from the instant a RF mode change request was received at the RTI Interface. This was well within specifications.

6. Summary & Lessons Learned

Using the power of C++ class inheritance, this code was written to be readable, reusable, and highly maintainable. Each main program (referring to the RTI Interface and the SWEG Interface) are written in a "generic" condition, which calls functions that every Federate Manager class will have. It is up to the derived Federate Manager class that is used to redefine those functions, if desired. In addition, the Object and Interaction classes make heavy use of inheritance which also lead to a highly structured and reusable interface.

Lessons Learned / recommendations

- Isolate different processes onto different processors if possible. On the ACETEF gateway, an SGI Onyx, several processes were running at once. Although the machine was dedicated to the test, it still made a drastic improvement in performance (latency) to separate the processes.

- The Emitter Monitor, though not a planned feature from the beginning, proved to be extremely worthwhile, both in debugging and in the test observation. It was simple to use, relatively simple to set up, and this method of observation (shared memory with a monitor program) will likely be used henceforth.

- Coding Teamwork. This code was not written by one person. Several engineers had a hand in it, each doing their part and agreeing on what to pass and how. Finding the right people with the right experience for constructing the component pieces was a great time-saver.

Author Biography

LAWRENCE A. CONRAD is an electrical engineer specializing in communications at the Naval Air Warfare Center, Patuxent River MD. He was the lead programmer for the ACETEF HLA Interface for the JADS-EW test and is currently working with the Chesapeake Test Range to expand their HLA capabilities.