Orbiter 2010P1 HLA Interface

Introduction

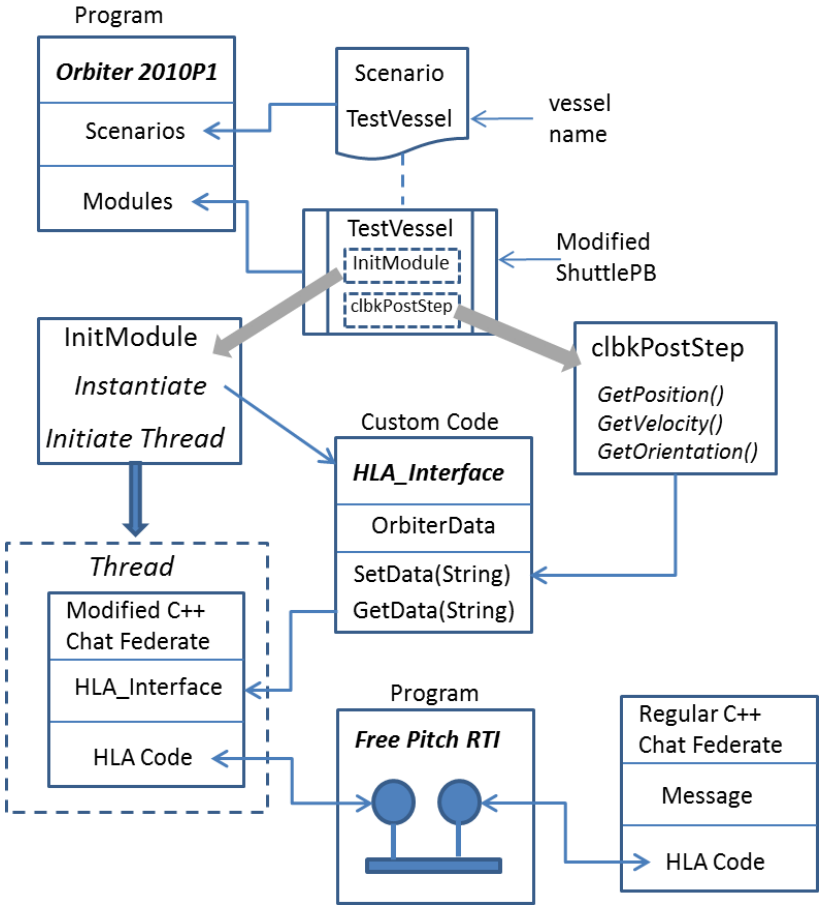
Orbiter 2010P1 is a powerful space flight simulator with built-in physics models for aerodynamics, atmosphere, orbital trajectories, docking, and ephemeris. Computer graphics functions include exterior views of a spac-craft, interior cockpit views with Multi-Function Displays (MFD) and Heads-Up Displays (HUD), planets, and landing bases. A world-wide community has created a vast variety of simulations for space exploration mission scenarios. This task plan provides technical details about the development of a High Level Architecture (HLA) interface that will enable the Simulation Smackdown and Orbiter communities to integrate existing orbiter simulations into a federation.

Scope

Three documents explain how to create a custom module, specify a vessel, and integrate the vessel into a scenario. The Orbiter Software Developers Kit (SDK) provides a C++ Application Programming Interface (API). Acrobat files within the subdirectory, *Orbiter2010\Orbitersdk\doc* , include API_Guide.pdf and API_Reference.pdf. The file API_Guide.pdf contains the Orbiter Programmer's Guide, which explains how to create a plug-in module that simulates a spacecraft. Section 1.1 explains module initialization and section 1.2 explains Vessel initialization. The file API_Reference.pdf contains the Orbiter API Reference manual. The Orbiter 2010 subdirectory, named Doc, includes a document named ScenarioEditor.pdf, which explains how to edit a scenario text file.

The Functional Flow Block Diagram (FFBD) depicts a test configuration with the Orbiter 2010P1 and free Pitch RTI programs. A scenario text file includes the name of a custom module named TestVessel, which is a modified version of the ShuttlePB module. New methods in the TestVessel module include InitModule and clbkPostStep. InitModule instantiates an object from a class named HLA interface.

The HLA_Interface class includes a buffer for data coming from TestVessel and it initiates a thread. The Thread contains a modified version of the C++ chat federate provided with the free Pitch RTI. The modified chat federate instantiates the HLA_Interface so that it can call the GetData method.



Functional Flow Block Diagram

The callback method named clbkPostStep is called by Orbiter after the state of the vessel changes. When Orbiter invokes this call back function, Vessel methods get the position, velocity, and orientation of the TestVessel. The clbkPostStep method calls the SetData method of the HLA_Interface to load the TestVessel data into the OrbiterData buffer.

Tasks

Developing an HLA Interface involves configuring the development environment, creating a module that defines a vessel and a scenario that creates the vessel, and creating a class that enables the exchange of data between two programs. Activities in the first task include configuring Visual Studio C++ to work with the Orbiter 2010P1 Application Programming Interface (API) and producing a new vessel with existing sample code. Modifications of the vessel module in task two will implement the InitModule and the post state change call back function. The InitModule will instantiate the HLA_Interface and spawn a thread for the modified C++ Chat federate. The call back function will get the current position, velocity, and orientation and set a data structure in the HLA Interface. A third task modifies the C++ Chat federate and conducts the test to determine whether data can get from Orbiter to a second Chat federate.

Task 1 Configure the development environment

Mohd Ali produced a tutorial video about configuring Visual Studio to develop modules for Orbiter 2010.
http://www.youtube.com/watch?v=VBRLkN__YIo&list=UUXA9ec1Npj308AY5r6vxALw&index=10

A discussion thread at the Orbiter Wiki provides some information about configuring Visual Studio to work with the Orbiter 2010 libraries.The section on Visual C++ 2010 Express references Mohd Ali's video. Advice for troubleshooting errors appears at the end of the article.
http://www.orbiterwiki.org/wiki/Free_Compiler_Setup

The ShuttlePB module defines a spacecraft with a relatively minimum amount of code. After compiling the ShuttlePB with a new name, like TestVessel, place the file in the appropriate directory and activate the scenario editor. The scenario editor is a plug-in that you can activate through the Modules tab of the Orbiter 2010P1 user interface. The Scenario Editor document explains how to edit a scenario text file to identify the TestVessel module by name. A scenario can instantiate a vessel in orbit.

Selecting the scenario will activate the TestVessel. Section 7.1 of the Orbiter User Manual provides a table of general keyboard commands. The F3 key invokes a pop-up menu for changing the focus from one spacecraft to another within a scenario. The F1 key toggles between the interior and exterior view of the spacecraft in the current focus.

Task 2 Revise a TestVessel module

Methods in the class TestVessel module include the Vessel initialization and exit routines and a callback function to set the vessel variable values. Efforts in this task involve creating methods for InitModule and a clbkPostStep. According to the Orbiter Programmer's Guide, the module initialization method is called once when the scenario loads. This initialization method shall initiate a thread to execute a modified version of the C++ Chat federate and instantiate an HLA_Interface class object. Task 3 will explain the modification of the Chat federate.

Section 8.52.1 of the API Reference provides a detailed description of the Vessel2 class, which includes the callback functions:

- virtual void clbkPreStep (double simt, double simdt, double mjd)
  Time step notification before state update.
- virtual void clbkPostStep (double simt, double simdt, double mjd)
  Time step notification after state update.

The callback function, clbkPostStep, executes after the state of the vessel updates. When Orbiter 2010 updates the position of the TestVessel, the simulation ought to call the clbkPostStep after updating the position. The Vessel class includes methods for getting the position, orientation, velocity and other system attributes. Custom code in this method will transfer the data to a structure within the HLA_Interface via a call to the SetData method.

Task 3 Modify C++ Chat Federate and Test the HLA Interface

The free Pitch RTI comes with C++ Chat federate in the samples folder. Activities associated with this task include modifying a copy of the C++ chat federate to instantiate an HLA_Interface class object, and changing the code that reads a message from the command line interface to use the GetData method of the HLA_Interface to populate the message. Other changes involve waiting for changes in the HLA_Interface OrbiterData. Later revisions will incorporate conservative time management and reference frame adjustments. For the initial version, the point is enable the C++ Chat federate to obtain a message from the HLA_Interface and convey it to another Chat federate.

Testing the HLA_Interface involves running the Pitch RTI and the Orbiter 2010P1 scenario that creates the Test Vessel. When Orbiter invokes the InitModule, the modified C++ Chat federate ought to appear on the Pitch RTI window. Manually start another Chat Federate, which will instantiate a second lollypop on the diagram in the Pitch RTI user interface. As the vessel orbits the Earth, the Chat federates ought to update with the position, velocity, and orientation data.

Deliverables

The C++ code for implementing a generic vessel and the HLA_Interface class are the custom code deliverables. Pitch Technologies produced the C++ Chat federate, so modifications are subject to the licensing agreement written into the code. Subsequent versions of the modified Chat federate can reduce the code to the bare-bones of receiving a message and sending the message; as other functions for time management and reference frame adjustments are added, the code will become unique enough to release with a HLA_Interface archive. The Orbit Hangar website is a repository for Orbiter 2010P1 addons. When the team determines the code is ready for public consumption, an archive can be uploaded to Orbit Hangar.

Resources

Programmers at Ajou University and the University of Alabama in Huntsville are currently, working on the HLA Interface development task. Only 25 miles separates Brunel University and University College London, where Professor Martin Schweiger teaches. Perhaps, students from Brunel can have face-to-face meetings with the creator of Orbiter 2010 and other members of the development team. Martin Tapp posted messages on the Orbiter Wiki, which can serve as a starting point for asking detailed technical questions to the Orbiter simulation development community.