CS582

Modeling and Simulation II

HLA LUNAR FEDERATE DEVELOPMENT AND INTEROPERABILITY DEMONSTRATION

ERIC BRIGHT, MICHAEL DUNNING, WILL GARRISON, ERIK MAYHAN,
BRIAN MCCORMICK, KONSTANTINOS PRAPIADIS,
JODY SMITH, HISHAM ZEINELABDIN

CS582 MODELING AND SIMULATION II
FINAL REPORT

INSTRUCTOR:

DR. MIKEL PETTY

APRIL 28, 2014

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ALABAMA IN HUNTSVILLE
HUNTSVILLE, AL 35899

ACKNOWLEDGMENTS

The following entities contributed support in various forms from donation of software, advisement, team coordination, and technical support, to monetary support. All of which was greatly appreciated and herein acknowledged for its significant impact on the learning experience. The project organization, technical support and team spirit provided by all NASA personnel provided an inspirational driving force to keep the project moving.

Aegis Technologies
Analytic Graphics, Inc.
Johnson Space Flight Center, NASA
Kennedy Space Flight Center, NASA
Marshall Space Flight Center, NASA
Mäk Technologies, Inc.
Pitch Technologies, Inc.
Simulation Interoperability Standards Organization (SISO)
Society for Modeling and Simulation International (SCS)

TABLE OF CONTENTS

Acknowledgments iii
List of Figures v
Abstract 1
Motivation and Description 1
  Scenario and Federate Rationale 2
    Lunar Communication Satellite Federate (LComSat) 3
    Lunar Mass Driver Federate (LMD) 3
    Cargo Transport Federate (CargoTrans) 4
Process and Project Management 5
Development 8
  Approach 9
  Lunar Communications Satellite Federate 14
  Lunar Mass Driver Federate 15
  Cargo Transport Federate 21
Discussion 34
Conclusions 34
References 36

LIST OF FIGURES

Figure 1. UAH project team collaboration web site 6
Figure 2. Project schedule for UAH team distributed simulation project 7
Figure 3. Lollipop chart of federates participating in the SEE 2014 lunar HLA federation 8
Figure 4. Screenshot of java publisher and subscriber skeleton federates.... **Error! Bookmark not defined.**
Figure 5. Class diagram of java publisher and subscriber skeletons**Error! Bookmark not defined.**
Figure 6. Sequence diagram of ownership transfer push from A to B**Error! Bookmark not defined.**
Figure 7. Sequence diagram of ownership transfer pull from B to A**Error! Bookmark not defined.**
Figure 8. State diagram of the cargo drone launch 23
Figure 9. State diagram of the cargo drone docking 24
Figure 10. State diagram for landing of the cargo drone..........26
Figure 11. Three dimensional model used for visualization of the cargo drone....27
Figure 12. Visualization of cargo drone landing 28
Figure 13. Visualization of 29
Figure 14 Visualization of 30
Figure 15. Visualization of 30
Figure 16. Visualization of 31
Figure 17. Mass driver visualization 32
Figure 18. Visualization of 33
Figure 19. Satellite constellation as seen from the L2 Outpost visualized in the Celestia package used by the Munich team..........34

LIST OF ACRONYMS

API	Application programming interface
CargoTrans	Cargo Transport
CRC	Central Runtime Component
FOM	Federation Object Model
HLA	High Level Architecture
LComSat	Lunar Communication Satellite
LMD	Lunar Mass Driver
LRC	Local Runtime Component
NASA	National Aeronautic and Space flight Administration
RTI	Run Time Infrastructure
SOM	Federate Object Model
STK	Systems Tool Kit

ABSTRACT

Stressing interoperability our project team reengineered the High Level Architecture (HLA) Lunar Communication Satellite (LComSat) and Lunar Mass Driver (LMD) simulation federates from the previous year's project to better serve the lunar colony. Within the simulation scenario a newly developed Cargo Transport (CargoTrans) federate managed drone cargo ships laden with LO_2 extracted from lunar regolith assisting with logistical resupply of a space station outpost at Earth-Moon Lagrange point 2. The CargoTrans federate obtained ownership of the drones as they were launched from the lunar surface with the mass driver. After directing their multi-day transit, the CargoTrans federate granted an ownership change to the outpost for docking and unloading. The CargoTrans federate then reacquired ownership for the return trip and lunar landing. The LComSat federate provided all required communication pathways needed for the lunar colony. Using satellite-to-satellite relay schemes, all surface and space locations were served including those occulted by the body of the moon.

DESCRIPTION AND MOTIVATION

Devised as a learning framework for teaching "real world" project management and teamwork for university modeling and simulation students, Dr. Mikel Petty organized this task as an exercise in a distributed simulation course he teaches each year at the University of Alabama in Huntsville. The goal of the project was to apply the High Level Architecture (HLA) distributed simulation standard [1] in the development of student defined simulations or federates. The student developed federates were required to interoperate with other teams' federates within a simulation or HLA federation of a futuristic lunar colony. Organized by NASA the lunar colony simulation brings together many other universities from around the world to participate in the distributed simulation. A demonstration of the developed simulation entitled the Simulation Exploration Experience (SEE) took place at a convention of the Simulation Interoperability Standards Organization (SISO) and the Society for Modeling and Simulation International (SCS) in April of 2014 in Tampa, Florida.

Bringing together nine university teams from the US and Europe, as listed in and roles their simulation , NASA organizers provided four months of assisted program management, technical advisement, and communication conduits for all the teams. Teams were allowed to exercise their individual federates through network access to servers at NASA all within the larger distributed federation. NASA provided federates running on their servers to access ground truth time and physical coordinates. The overall simulation scenario defined by NASA was of a lunar colony physically located in the Aitken Basin on the southern far side of the moon with the exception of a space station outpost at

Table 1 List of University Teams and Simulations

University Teams	Simulation
Munich University	L2 Outpost
Genoa University	Communications
University of Bordeaux	Supply Depot
Brunel/Exeter Universities	Oxygen Factory
Penn State University	Asteroid Capture, Lunar Lander
University of Nebraska	Cargo Rover
University of AL in Huntsville	LComSat, LMD, CargoTrans
University of Calabria	Asteroid Defense
California Polytechnic University	Space Elevator

Earth-Moon Lagrange point 2. Building upon prior year efforts, each team defined the rationale, entities, and roles their simulation played in the larger scenario, and then through negotiation worked out a scenario for mutual accomplishment.

In addition to navigating the dynamics of the class project making optimal use of team member skill sets to define realizable goals, the UAH team had to negotiate and devise interface requirements and implementations for interoperating with the other teams. Within the framework of the HLA interface standard, all object, attribute, and sequence interchange between team federates had to be defined. The lack of discrete detail definition within the HLA standard allows much flexibility in how these interfaces are defined but subsequently requires up-front negotiation to map them out.

SCENARIO AND FEDERATE RATIONALE

As a prospective path for manned space missions in the future, NASA developed the overall scenario to include a lunar colony and allow the university teams to use their creativity and rational reasoning to develop potential systems of benefit. Through the paradigm of distributed simulation, each team can explore the pro's and con's of their systems including the needed interoperability with other systems to make them function in the greater mission. As in the real world, the decision to include a system does not simply come from how well that system performs its individual task but from how well it functions in producing the goals of the greater mission.

While the physical location of the lunar colony is defined by NASA to be in the Aitken Basin on the southern far side of the moon, it does not preclude the existence of space borne platforms that may be used in service to the colony or in the mission of the colony. In addition other exploratory locations on the lunar surface may be used for their particular location or features. Case-in-point, Hadley Rille, the site of the Apollo 15 landing, is on the near side of the moon and provides a popular point for continuous communication with the earth.

Previous teams created additions to the scenario for the harvest of regolith from the surface for extraction of oxygen. The oxygen can be used for breathable atmosphere, as well as, for propulsion. This oxygen can be used for activities on the lunar surface, for space missions beyond the moon, for perpetual space stations, and for platforms in earth orbit. Due to the low gravity of the moon relative to that of the earth, it requires less energy to transport resources from the moon to earth orbit than from the earth's surface.

A space station outpost at Lagrange point 2 of the Earth-Moon system was included by one team from previous years which could serve as a destination for beyond the moon space mission docking. Through group discussions the current UAH team developed a mission and rationale that could make use of existing federates and create value added to the overall scenario. The communication satellite and mass driver federates developed by previous teams from this university provided some of the initial infrastructure for resupply of the L2 Outpost from the lunar surface. With the addition of a federate to manage the space flight and docking of cargo drones, the mass driver could launch drones full of LO_2 from the lunar surface and the satellite constellation provide the communication conduit to get them to their destination.

LUNAR COMMUNICATION SATELLITE FEDERATE (LCOMSAT)

Line-of-sight lunar/terrestrial communications becomes impossible for a lunar base on the back side of the moon in Aitken Basin. Repeaters on the lunar surface are not a solution as there are no ionized atmospheric layers to bounce signals long distances resulting in an impractical number of repeater towers. A lunar satellite constellation can provide lunar/lunar and lunar/terrestrial communications paths, as well as, repeating capability for outposts or ships in space blocked or occulted by the moon.

LUNAR MASS DRIVER FEDERATE (LMD)

With payload launch from the lunar surface requiring approximately 5% of the energy required for terrestrial launch, a lunar mass driver provides an attractive launch mechanism for the distribution of lunar products to space stations, interplanetary space ships, and even earth orbiting satellites. Using a superconducting quench gun powered by either solar or nuclear power, the mass driver can propel cargo drones at up to 1700 m/s to escape the moons gravity. With no atmospheric resistance effects to interfere, the drones only require the structural protection of a reusable carriage while accelerating along the mass driver track. The lack of convective heat transfer and the cold temperatures of the moon surface allow the efficient refrigeration needed for the superconducting magnets and the static storage of energy within the mass driver track.

For the current scenario the LMD federate launches cargo drones filled with LO_2 harvested from the lunar regolith for use by any space port within reach. Ownership of the position and velocity attributes of the drone are acquired by the LMD which then uses an internal physics model to accelerate the drone to the end of the track. Ownership of all drone attributes is then transferred to the Cargo Transport Federate for logistical distribution to the drone's destination

CARGO TRANSPORT FEDERATE (CARGOTRANS)

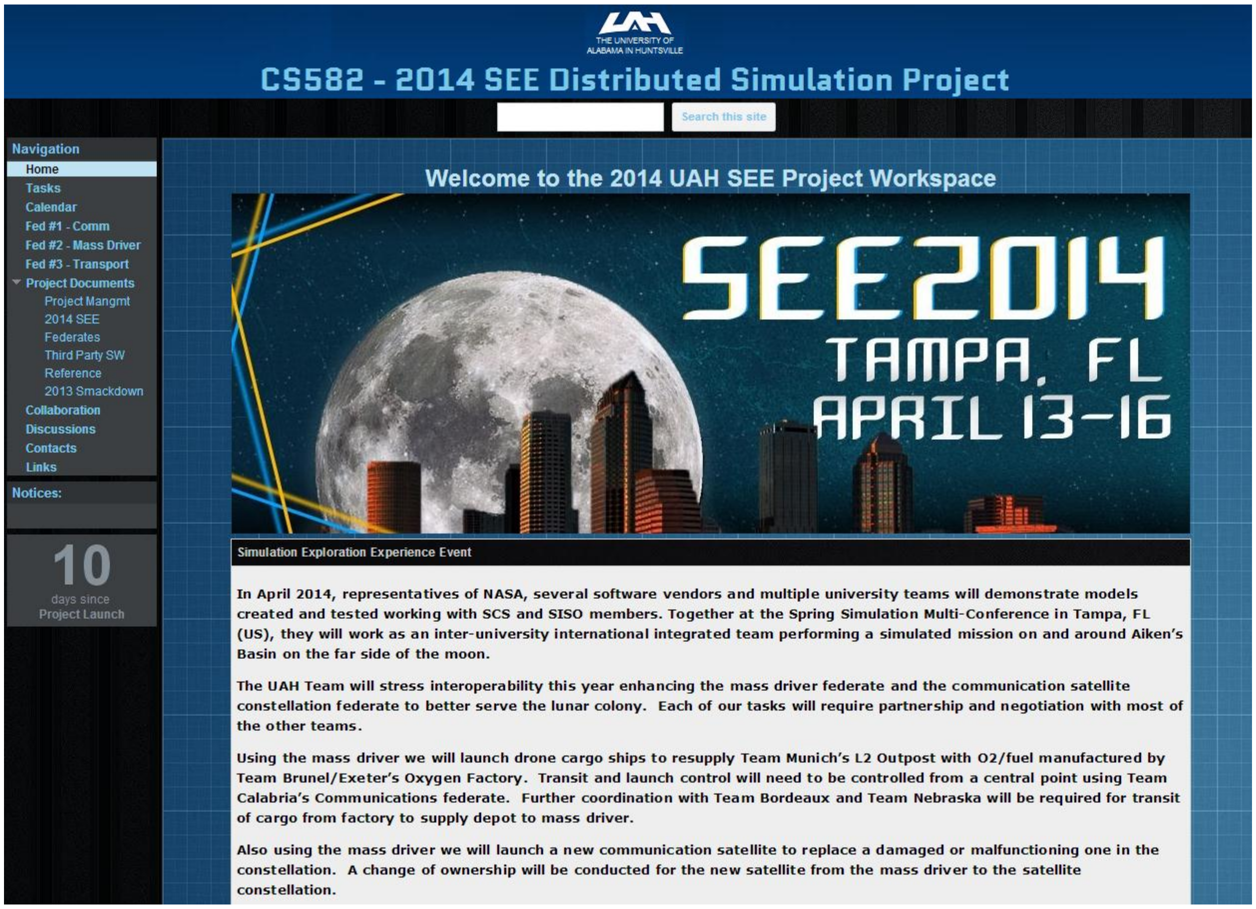
The Cargo Transport federate was created to create a constructive narrative for the SEE event. The Earth-Moon Lagrange Point station created by the Munich University team would serve as an ideal propellant depot for deeper exploration of the solar system. This propellant depot would be capable of fueling departure stages in orbit around the Lagrange Point, both reducing the launch cost from Earth, and providing a low gravity point of departure. To provide the fuel for the station, lunar regolith could be mined to extract oxygen and hydrogen. The Cargo Transport federate fills the capability gap of transporting resources from the surface of the moon to the station.

PROCESS AND PROJECT MANAGEMENT

In preparation for the task, our instructor provided several weeks of lectures on distributed simulation and the High Level Architecture standard. Additional instructional tutorials by Björn Möller of Pitch Technologies on creating HLA federates were provided through video conferences with the NASA organizers. A team leader was chosen that became responsible for the project task exactly as if it were a task in industry. A statement of work was generated along with a schedule in which both made optimal use of available resources. Due to a limited number of individuals with software writing skills, these individuals had to provide their services for all three federates developed. The remainder of the team was divided by federate to provide technical modeling and scenario development for each. One team member provided infrastructure support by investigating and exploiting all third party tools needed. Students from prior years were invited in to advise on approach and potential pitfalls.

Starting with the previous project team's software, the initial goal was to demonstrate its execution on the current laboratory computers then followed with execution through the NASA servers. Using donated HLA runtime infrastructure packages from both Pitch Technologies and Mäk Technologies and VPN connections to the NASA servers, our team first exercised the example federates provided with the RTI packages. Then recompiling the previous team's software with up to date IP addresses, a baseline running federation was created to learn from. Building upon the existing federates, the decision was made to stress interoperability with the other project teams and make use of those mechanisms within the HLA standard that would best support that. Monitoring the NASA tech forum provided quick access to common issues that all the teams faced especially those that involved mutual interactions. A project team website, shown in Figure 1, was created to allow a common site for communication and archive of all programmatic materials. Additional shared cloud drive space provided a common location for archive developed software.

The task schedule, created in Microsoft Project and shown in Figure 2, provided reminders of those accomplishments and milestone needed before the project would be ready for the demonstration event. With school spring break and semester schedule impacts, the project team worked straight through spring break in order to meet the deadline.



Simulation Exploration Experience Event

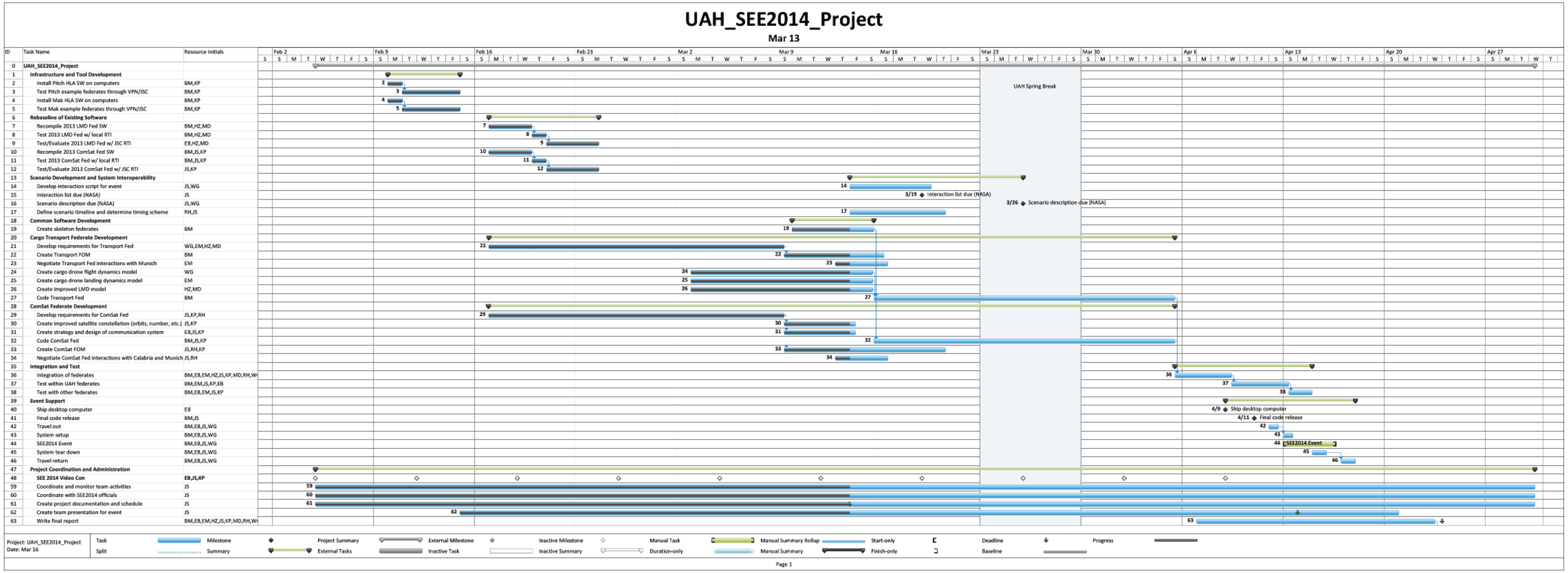
In April 2014, representatives of NASA, several software vendors and multiple university teams will demonstrate models created and tested working with SCS and SISO members. Together at the Spring Simulation Multi-Conference in Tampa, FL (US), they will work as an inter-university international integrated team performing a simulated mission on and around Aiken's Basin on the far side of the moon.

The UAH Team will stress interoperability this year enhancing the mass driver federate and the communication satellite constellation federate to better serve the lunar colony. Each of our tasks will require partnership and negotiation with most of the other teams.

Using the mass driver we will launch drone cargo ships to resupply Team Munich's L2 Outpost with O2/fuel manufactured by Team Brunel/Exeter's Oxygen Factory. Transit and launch control will need to be controlled from a central point using Team Calabria's Communications federate. Further coordination with Team Bordeaux and Team Nebraska will be required for transit of cargo from factory to supply depot to mass driver.

Also using the mass driver we will launch a new communication satellite to replace a damaged or malfunctioning one in the constellation. A change of ownership will be conducted for the new satellite from the mass driver to the satellite constellation.

Figure 1. UAH project team collaboration web site



ID	Task Name	Resource Initials
0	**UAH_SEE2014_Project**	
1	**Infrastructure and Tool Development**	
2	Install Pitch HLA SW on computers	BM,KP
3	Test Pitch example federates through VPN/JSC	BM,KP
4	Install Mak HLA SW on computers	BM,KP
5	Test Mak example federates through VPN/JSC	BM,KP
6	**Rebaseline of Existing Software**	
7	Recompile 2013 LMD Fed SW	BM,HZ,MD
8	Test 2013 LMD Fed w/ local RTI	BM,HZ,MD
9	Test/Evaluate 2013 LMD Fed w/ JSC RTI	EB,HZ,MD
10	Recompile 2013 ComSat Fed SW	BM,JS,KP
11	Test 2013 ComSat Fed w/ local RTI	BM,JS,KP
12	Test/Evaluate 2013 ComSat Fed w/ JSC RTI	JS,KP
13	**Scenario Development and System Interoperability**	
14	Develop interaction script for event	JS,WG
15	Interaction list due (NASA)	JS
16	Scenario description due (NASA)	JS,WG
17	Define scenario timeline and determine timing scheme	RH,JS
18	**Common Software Development**	
19	Create skeleton federates	BM
20	**Cargo Transport Federate Development**	
21	Develop requirements for Transport Fed	WG,EM,HZ,MD
22	Create Transport FOM	BM
23	Negotiate Transport Fed interactions with Munich	EM
24	Create cargo drone flight dynamics model	WG
25	Create cargo drone landing dynamics model	EM
26	Create improved LMD model	HZ,MD
27	Code Transport Fed	BM
28	**ComSat Federate Development**	
29	Develop requirements for ComSat Fed	JS,KP,RH
30	Create improved satellite constellation (orbits, number, etc.)	JS,KP
31	Create strategy and design of communication system	EB,JS,KP
32	Code ComSat Fed	BM,JS,KP
33	Create ComSat FOM	JS,RH,KP
34	Negotiate ComSat Fed interactions with Calabria and Munich	JS,RH
35	**Integration and Test**	
36	Integration of federates	BM,EB,EM,HZ,JS,KP,MD,RH,W
37	Test within UAH federates	BM,EM,JS,KP,EB
38	Test with other federates	BM,EB,EM,JS,KP
39	**Event Support**	
40	Ship desktop computer	EB
41	Final code release	BM,JS
42	Travel out	BM,EB,JS,WG
43	System setup	BM,EB,JS,WG
44	SEE2014 Event	BM,EB,JS,WG
45	System tear down	BM,EB,JS,WG
46	Travel return	BM,EB,JS,WG
47	**Project Coordination and Administration**	
48	**SEE 2014 Video Con**	**EB,JS,KP**
59	Coordinate and monitor team activities	JS
60	Coordinate with SEE2014 officials	JS
61	Create project documentation and schedule	JS
62	Create team presentation for event	JS
63	Write final report	BM,EB,EM,HZ,JS,KP,MD,RH,W

Task		Project Summary		Inactive Milestone	◇	Manual Summary Rollup	
Split		External Tasks		Inactive Summary		Manual Summary	
Milestone	◆	External Milestone	◆	Manual Task		Start-only	[
Summary		Inactive Task		Duration-only		Finish-only	]
Deadline	⬇	Progress					

Figure 2. Project schedule for UAH team distributed simulation project

DEVELOPMENT

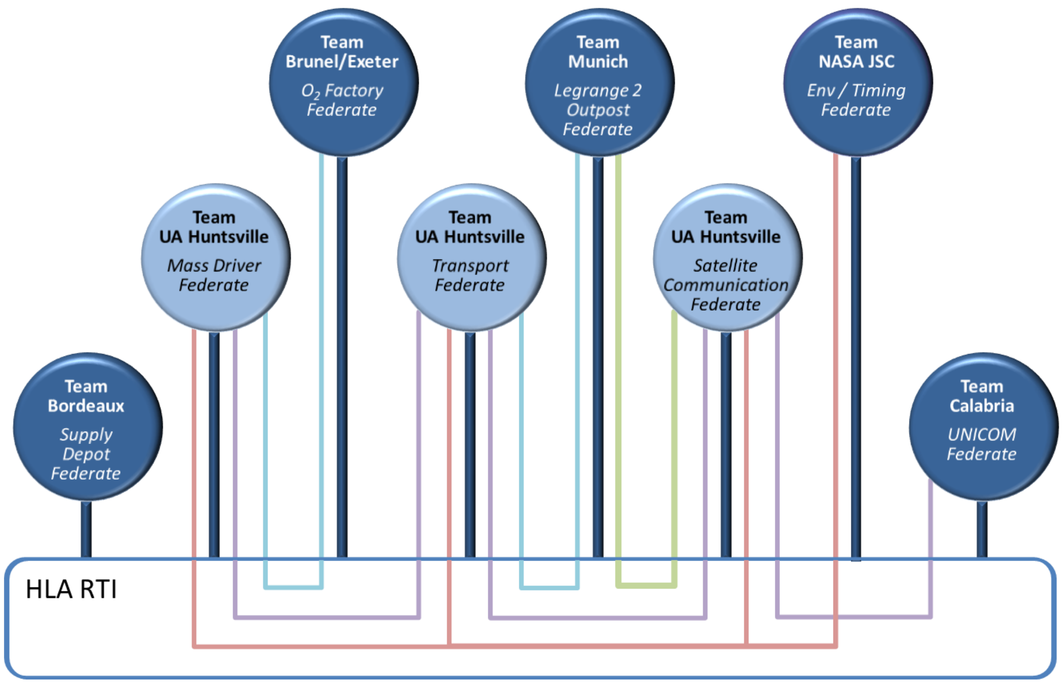


Figure 3. Lollipop chart of federates participating in the SEE 2014 lunar HLA federation

APPROACH

Providing a legacy to future teams, the current team developed generic skeleton federate software to which specific federate functionality could be added. All existing federates were reengineered and the new federates developed using the skeleton software.

SKELETON FEDERATES

Skeleton HLA federates were developed in Java and C++. Future HLA developers are encouraged to use these skeleton federates as a starting point for developing new federates, as well as, to improve them. Even if future teams just improve existing federates, we encourage them to look at the structure of the skeleton federates to get a better understanding of how an HLA federate is structured. The skeleton federates are packaged with the fully completed federate software developed for this task.

The Java skeleton federates were created using Eclipse, while the C++ skeleton federate was created using Visual Studio 2010 Professional Edition. The skeletons should run in other development environments such as NetBeans or Qt Creator after the expected routine changes for such things. Also, the skeletons were all developed using the Pitch RTI. We encourage future developers to finish what we started by getting the skeleton federates to work with the MäK RTI as well.

Java Skeleton Federates

There are two Java skeleton federates. The publisher Java skeleton creates a simple Java Swing front-end for a sample HLA federate. Its purpose is to publish sample cargo drone or entity data and perform a negotiated ownership transfer. The subscriber Java skeleton also creates a simple Java Swing front-end for a sample HLA federate. Its purpose is to subscribe to and receive the sample cargo drone data and perform an intrusive ownership pull.

To run the Java skeletons, first connect to NASA using SonicWALL's *NetExtender*. The download for *NetExtender* was free upon registration. We had four user accounts with each account having its own IP address and password, but that could change in the future so you should check with the system administrator at NASA to correctly log in. Generally, to run the Java skeleton federates you need to setup your Pitch CRC settings so that they correspond to the NASA account IP address that you are logged in to.

Next, you need to start Eclipse and import each Java skeleton project. After the projects are imported, they can be run by right clicking on the project folder in the Project Explorer window and selecting Run As… -> Java Application. See Figure 4 for a screen shot of these rather elementary GUI

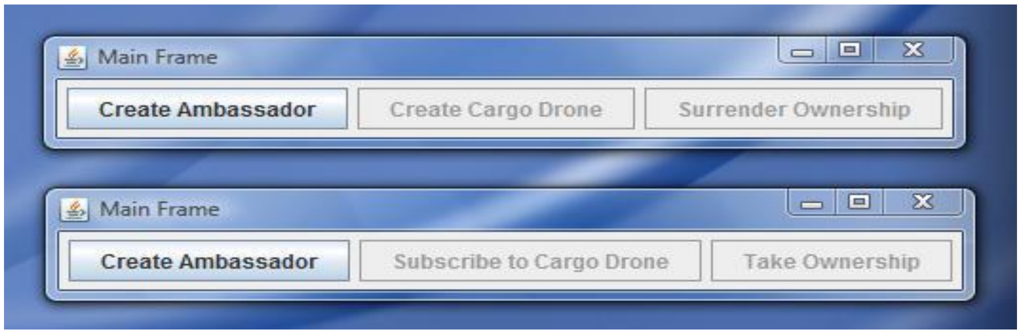


Figure 4. Screenshot of *Java Publisher* and *Subscriber* skeleton federates

applications. Take note of the console output windows in Eclipse - they give valuable information such as connection status not seen on the GUI's.

Java Publisher Skeleton Federate

For the *Java Publisher* skeleton, clicking the Create Ambassador button makes the application join the existing HLA federation. In order to simplify the skeleton code, the federation name ("SEE 2014"), the NASA Pitch RTI server location (192.168.15.2), and the list of FOM's were hard-coded in a static method called **createFederateAmbassador()**. When running through the NASA servers, verify that these items are consistent with those on the servers. The **createFederateAmbassador()** method is found in the *FederateAmbassador* class shown in the class diagram of **Error! Reference source not found.**. Note both Java skeletons share the same code base and thus have the same class organization. Close study of the *FederateAmbassador* constructor gives an idea on how time management works in an HLA application. Within these Java skeletons, time is not regulated but it is constrained.

After the Create Ambassador button is clicked, the Create Cargo Drone button is enabled. Clicking the Create Cargo Drone button creates a cargo drone object IAW the *PhysicalEntityTest* definition in the `SISO_Smackdown_2013_cargodrone.xml` FOM. The Java skeleton then begins publishing its name and mass attributes on a periodic basis via the *CargoDrone* class method called **postAttributes**(). Examination of this method can serve as a guide on transmitting various types of data.

Note the cargo drone simulation at each time step merely increments the mass attribute value by 1.0. This allows federates receiving this attribute to know that they are receiving the data correctly. Also note the simulation proceeds by the federate asking for a time advance and waiting until it is granted via the Pitch RTI **timeAdvanceGrant()** callback, doing the required work for that time interval, asking for another time advance grant, and continuing in this manner until done.

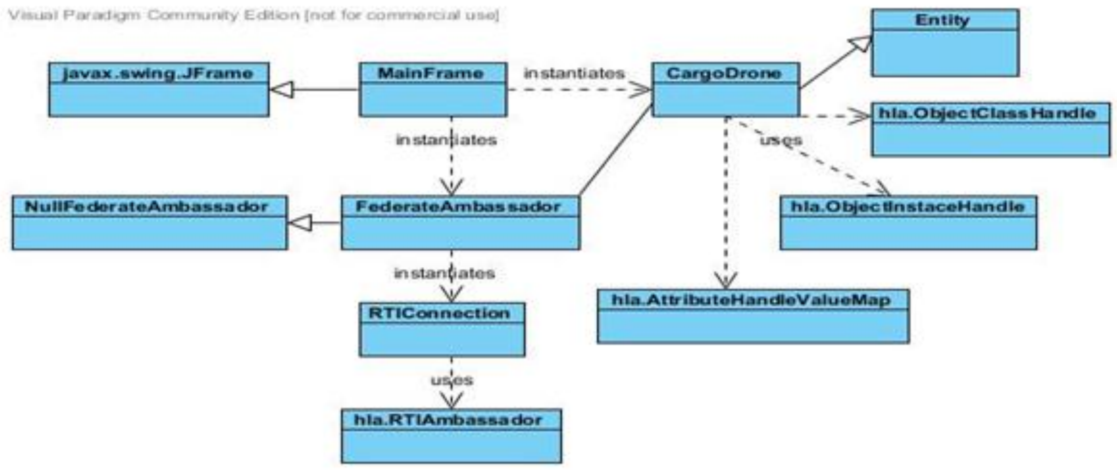


Figure 5. Class diagram of *Java Publisher* and *Subscriber* Skeletons

After the Create Cargo Drone button is clicked, the Surrender Ownership button in enabled. Clicking the Surrender Ownership button attempts to do a negotiated ownership transfer for the name and mass attributes of its cargo drone. If any federate agrees to take the ownership, the now so-called *Java Publisher Skeleton* releases the ownership of the cargo drone attributes and does nothing else. See **Error! Reference source not found.** below for the sequence diagram showing the steps of the ownership transfer push process.

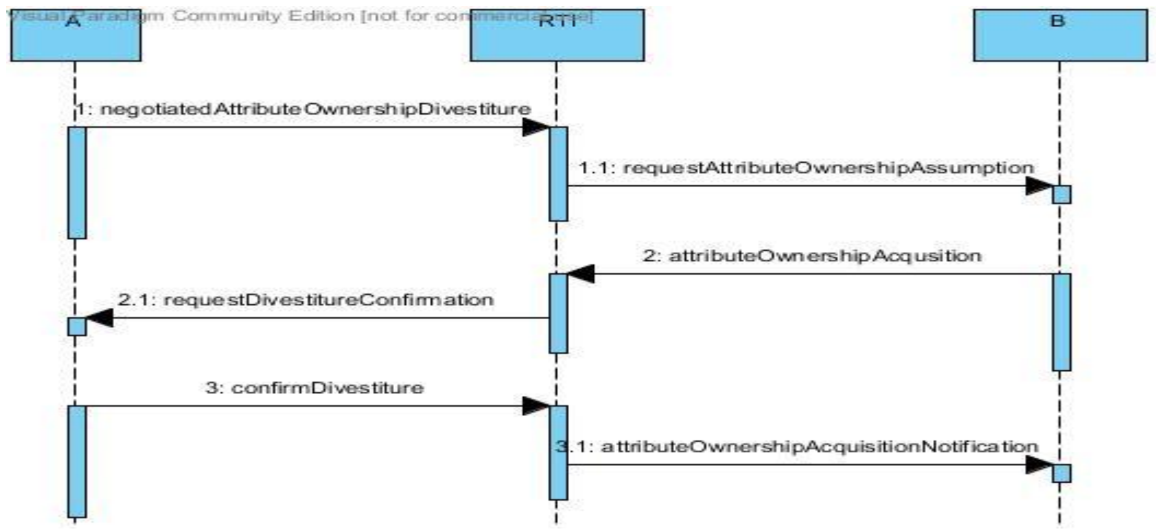

Visual Paradigm Community Edition [not for commercial use]
A
RTI
B
1: negotiatedAttributeOwnershipDivestiture
1.1: requestAttributeOwnershipAssumption
2: attributeOwnershipAcqusition
2.1: requestDivestitureConfirmation
3: confirmDivestiture
3.1: attributeOwnershipAcquisitionNotification

Figure 6. Sequence diagram of ownership transfer push from A to B

Java Subscriber Skeleton Federate

The same setup rules for running the *Java Publisher* also apply for the *Java Subscriber*. For the *Java Subscriber*, clicking the Create Ambassador does exactly what it does for the *Java Publisher*. The nice thing is that since both Java skeletons use the same code base, getting the *Java Publisher* correct means the *Java Subscriber* is also automatically correct.

Clicking the Create Ambassador button enables the Subscribe to Cargo Drone button. The Subscribe to Cargo Drone button makes the Java skeleton express an interest in receiving cargo drone name and mass attributes IAW the *PhysicalEntityTest* definition in the `SISO_Smackdown_2013_cargodrone.xml` FOM. If the cargo drone already exists, the Java subscriber will receive the cargo drone object via a callback from the Pitch RTI called **discoverObjectInstance()**. If the attributes are being published, it will receive callbacks from the Pitch RTI in a method called **reflectAttributeValues**(). You should look at this method as a guide for receiving various types of data in a Java HLA federate.

After the Subscribe to Cargo Drone button is clicked, the Take Ownership button in enabled. Clicking the Take Ownership button initiates a request to take the ownership of the name and mass attributes for a cargo drone owned by another federate as defined in the previously mentioned FOM. If any federate agrees to give up its ownership, the subscriber federate then turns into a publisher and begins publishing the name and mass attributes for the entity or cargo drone. See **Error! Reference source not found.** below for the sequence diagram showing the steps of the ownership pull process.

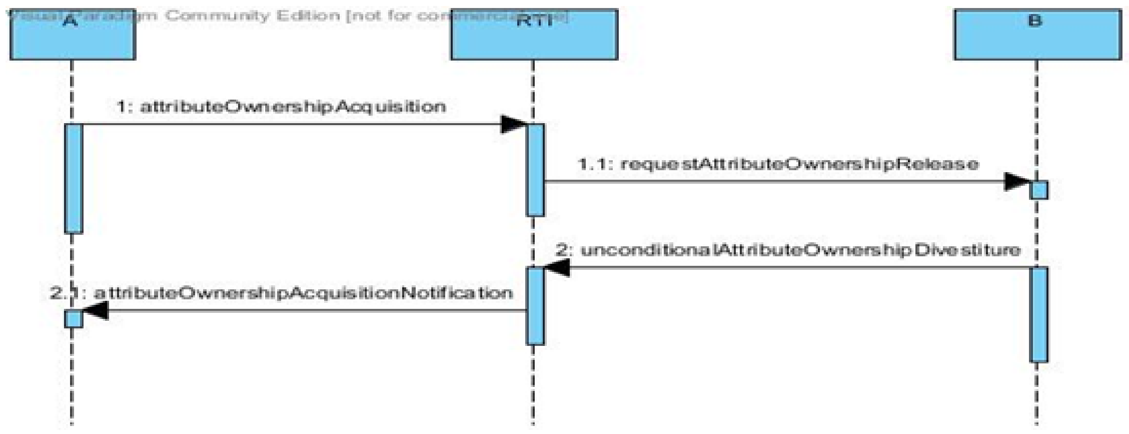

Visual Paradigm Community Edition [not for commercial use]
A
RTI
B
1: attributeOwnershipAcquisition
1.1: requestAttributeOwnershipRelease
2: unconditionalAttributeOwnershipDivestiture
2.1: attributeOwnershipAcquisitionNotification

Figure 7. Sequence diagram of ownership transfer pull from B to A

C++ Skeleton Federate

It is interesting to compare the structure of the Java skeletons to the C++ skeleton. Understanding the architecture of the Java skeletons allows quick assimilation of the C++ skeleton, and vice-versa. The C++ skeleton was developed with its expected use in mind. Its expected use is that it would receive and divest ownership of the name and mass attributes of the cargo drone defined in the aforementioned Java skeleton FOM.

The primary difference between the class hierarchy for the Java and C++ skeletons is that there is no *Entity* class in the C++ skeleton. We originally had an *Entity* class in the C++ skeleton, but it was removed while chasing down a bug that appeared to be from the use of pure virtual methods. We later found out the source of the bug was something else, but never added the *Entity* class back. Another difference is the Java *FederateAmbassador* class was renamed to *MyFederateAmbassador* in C++ to avoid a name conflict. Last, the Java skeletons are GUIs based on Java Swing, while the C++ skeleton is a Visual Studio console application.

The same NASA connection must be made to run the C++ skeleton. The project is easily loaded into Visual Studio 2010 Professional Edition. Running it actually involves a strange step. The project is loosely based on the Pitch RTI chat federate. So after compiling the project, we had to copy the resulting executable file over to the directory containing the installed Pitch RTI chat federate.

Over there, we made an exact copy of `chatcc1516e_vc100.bat` and renamed it to make it our own. Next we changed the executable that it was pointing to in that file (`.\chatcc1516e_vc100.exe`) to our own executable name. Future teams are encouraged to get the C++ skeleton to run from Visual Studio directly as we never had the time to figure out why we needed to perform this strange step. All the other processing in the C++ skeleton is the same as the Java skeletons and will not be regurgitated here. The only other changes were due to syntax differences of the languages.

An interesting thing about the C++ skeleton is that it can be used with the Java publisher skeleton. The Java publisher skeleton can negotiate an ownership transfer of the name and mass attributes of its cargo drone. The C++ skeleton can receive the ownership of the attributes. Getting the ownership to transfer between the C++ and Java federates was very satisfying and showed the true power of HLA interoperability.

However, the C++ skeleton does not actually publish the attributes for which it takes ownership. This seems very strange to us that a federate merely has to indicate that it can publish attributes in order to be able to obtain ownership, but not actually have to do it.

The C++ skeleton (and likewise the Java skeletons) never demonstrate an HLA interaction. However, you can easily find a Java HLA interaction example in the CargoTrans federate. It can easily be moved into the Java skeletons for any future teams that are so inclined.

LUNAR COMMUNICATIONS SATELLITE FEDERATE

The number of satellites needed and their relative inclinations for good coverage were determined using Analytic Graphics Incorporated’s System Tool Kit (STK) software. Priority coverage was given for lunar surface locations in Aitken Basin and Hadley Rille. Frozen orbits were chosen to minimize station keeping fuel usage; however, lunar satellites are heavily affected by third bodies and considerable changes in the gravity field of the moon.

Each communication activity is routed through the satellite constellation using line-of-sight calculations from sender, receiver and necessary acquired satellites. Interactions are processed with the Radio federate on the NASA server and the other team’s communication federates.

The 2014 UAHuntsville team decided to use the existing LCANSat2013 federate, which is an integration of the LCANServ, LCANSat, and 3D satellite constellation visualization federations that were developed by the 2012 UAHuntsville team.

The main functions of this federate are the propagation of constellations of satellites orbiting the moon, providing inter-communication capabilities between the physical entities in the mission and visualization of the satellites constellation and all the other physical entities in the federation.

This federate was built using the Eclipse Java development environment, and AGI’s Systems Tool Kit (STK) *Components for Java*. More specifically, the Dynamic Geometry Library (DGL) was used for propagation of the satellites and Insight3D for the visualization of the satellites.

The federate propagates and renders a 6-satellite constellation. The satellites are members of the *PhysicalEntity* object class defined in the `SISO_Smackdown_2013_Entity` FOM. Once the federate was granted time advance by the Environment federate, each satellite published its logical time, entity name, reference frame and most up-to-date Cartesian coordinate position. Other federates which were interested in obtaining the attributes of satellites could subscribe to and receive the updates sent by the federate.

The second module is the communication server that is used to facilitate lunar communications among different federates. The communication is only possible when there is a line of sight between the two objects. In order to calculate line-of-sight information, the federate subscribes to the name, reference frame and position attributes of the *PhysicalEntity* object class and maintains each entity’s latest position.

We also fixed some problems we found during the testing. When a federate resigned from the federation, the callback method **removeObjectInstance()** is called in order to tell the insight3D viewer to remove the marker and text representing the physical entity of that resigned federate. This would cause the Insight3D viewer to crash. We surrounded the errant code with a try-catch block in order to prevent the entire Insight3D viewer from crashing during the event.

LUNAR MASS DRIVER FEDERATE

The Lunar Mass Driver (LMD) federate used in the 2013 event carries over from last year in name only. The LMD instituted last year relies on a custom spacecraft designed by the 2013 team implemented as an add-on vessel within the freeware space simulator Orbiter 2010. This vessel contained an interface programed in Microsoft Visual Studio 2010 C++ to connect outputs from the vessel to the rest of the distributed simulation. After hearing presentations and descriptions of the 2013 federate, the initial idea was to reuse the existing code and improve the physics model used within the simulation. Also decided was to implement the first-ever ownership transfer demonstration held at the event using the LMD with a Cargo Drone object as a payload. Unfortunately, after spending several weeks working with Orbiter 2010 and attempting to run last year's code, improving the physics-model within Orbiter while implementing an ownership-transfer after launch proved to be completely impractical. As such, an entirely new physics-based LMD federate was programmed in C++ and used to take ownership of a Cargo Drone instance from the Cargo Transport federate, launch it on command, and return ownership after launch is complete.

Initial Work

The LMD used by the previous team was originally developed as a free add-on from the Orbiter repository. It was written by Yuri Kulchitsky for Orbiter 2006 and includes an LMD model, a base on the moon, and a scenario file which loads an example mission into Orbiter. The last team implemented this scenario into the federation with virtually no changes, save moving the base to the appropriate Aitkin Basin location on the moon. Several members of the previous team discussed the event and suggested two things. First, that the simulation appeared "jerky" during the actual event and that render times seemed more like a slideshow. Second, that the implementation of HLA might be better handled by making use of another free add-on for Orbiter called OrbConnect, which allows data to be output from objects in Orbiter to a chat window. Thus, the 2014 team decided to reuse the previous federate, but make improvements to it. The first goal was to improve the linear acceleration physics model to one using non-linear acceleration, or jerk, and the second goal was to reduce the Orbiter LMD track length to something more realistic for lunar construction.

The LMD model included in the scenario written by Yuri Kulchitsky purports to use a linear acceleration model to achieve lunar escape velocity. The documentation included with the model states that the track is 108 kilometers in length and the output speed by default is 2,000 meters per second. Investigations within Orbiter proved this to be the case. When using a linear acceleration model along with a NASA "Man-rated" maximum of 3G acceleration, one does indeed need about 103 kilometers to achieve lunar escape velocity. Unfortunately, the International Space Station, the largest man-made object ever assembled in space, features a main truss only about 110 meters in length. The idea of building anything on the moon over 62 miles long in the timeframe suggested by the competition outline was therefore deemed grossly inaccurate. Again, using the ISS as a baseline, a track roughly 20 times as long was considered more realistic. Given the fact that it would be built on the lunar surface and not in free space, this 2,000 meter track would balance meaningful additional velocity to orbit while still being feasible to construct. The new LMD track is visually shown in **Error! Reference source not found.** using Orbiter 2010.

In order to improve the constant, i.e. linear, physics model used by default for the Orbiter LMD, it

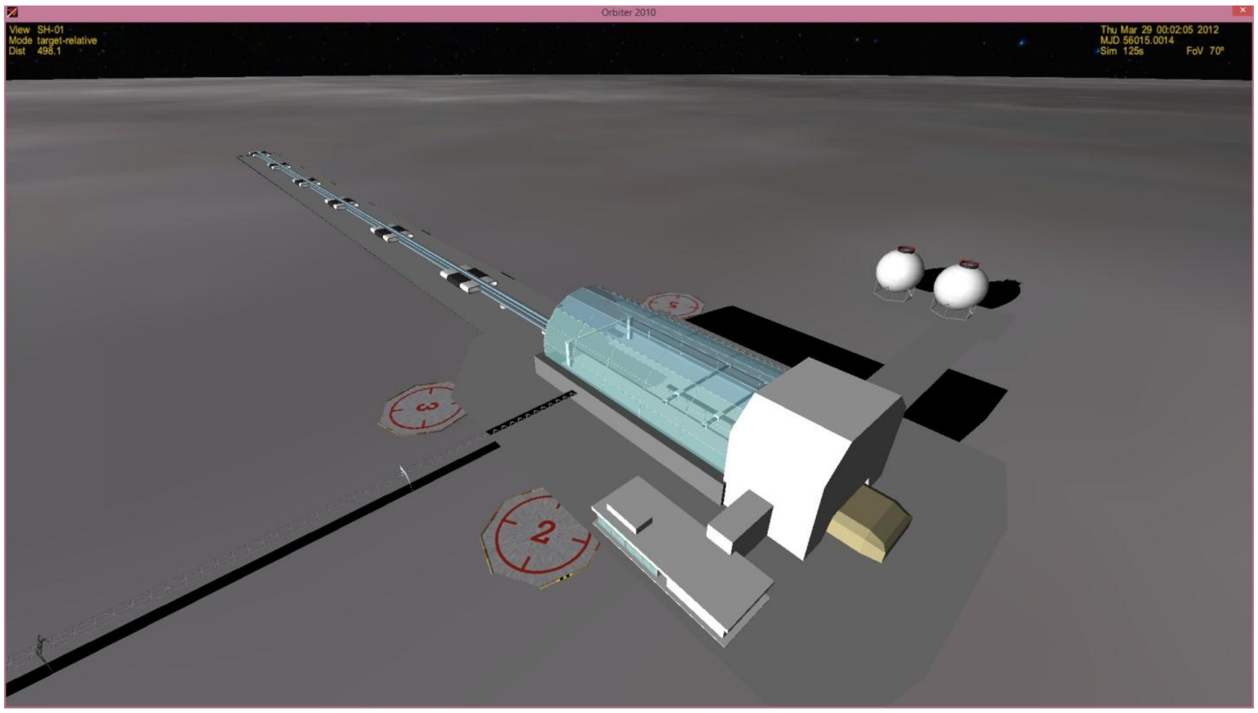


Figure 8. New 2014 LMD model with shortened 2,000 meter track

was decided to add a more realistic non-linear acceleration into the model. This is called Jerk, and is representative of what happens as you press the accelerator pedal down while pulling away from a stop light. The equation for position at time *t* is given by:

$$x(t) = x_0 + v_0\Delta t + \frac{1}{2}a\Delta t^2$$

While the equation for position at time t due to non-linear acceleration is given by:

$$x(t) = x_0 + v_0\Delta t + \frac{1}{2}a\Delta t^2 + \frac{1}{6}j\Delta t^3$$

A small program was written in C++ to test both linear and non-linear acceleration as to determine what sort of velocity gain (ΔV) could be expected from various configurations and input parameters. The inputs decided as critical were track length and maximum acceleration that the payload could endure. This would either be set by human physiological constraints or structural limitations of the payload.

After deciding on a 2,000 meter track and the physics equations that would be programmed to drive it, work began in earnest to learn the API and add-on creation process necessary to work in Orbiter 2010. Meshlab was used along with a plugin available from the Orbiter repository to import and export the native ".msh" file that Orbiter uses for models. The LMD model was imported and changed to reduce

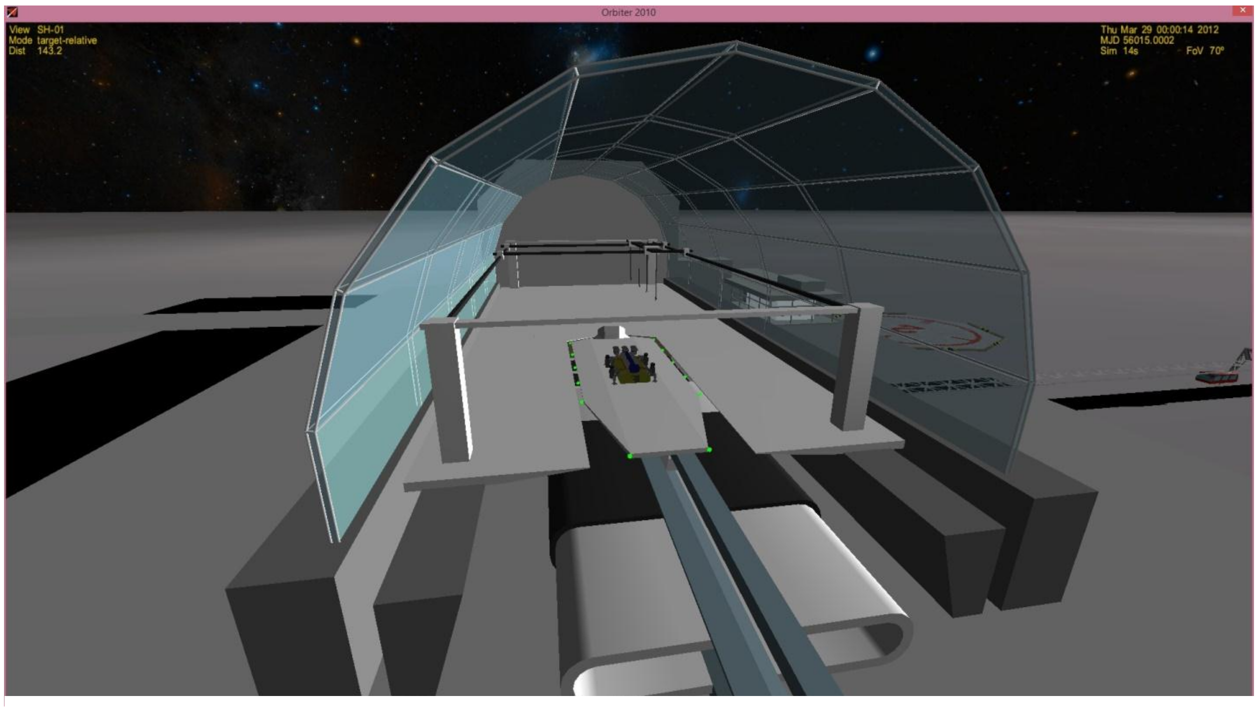


Figure 9. 2014 UAH Cargo Drone model loaded onto the new 2014 UAH LMD

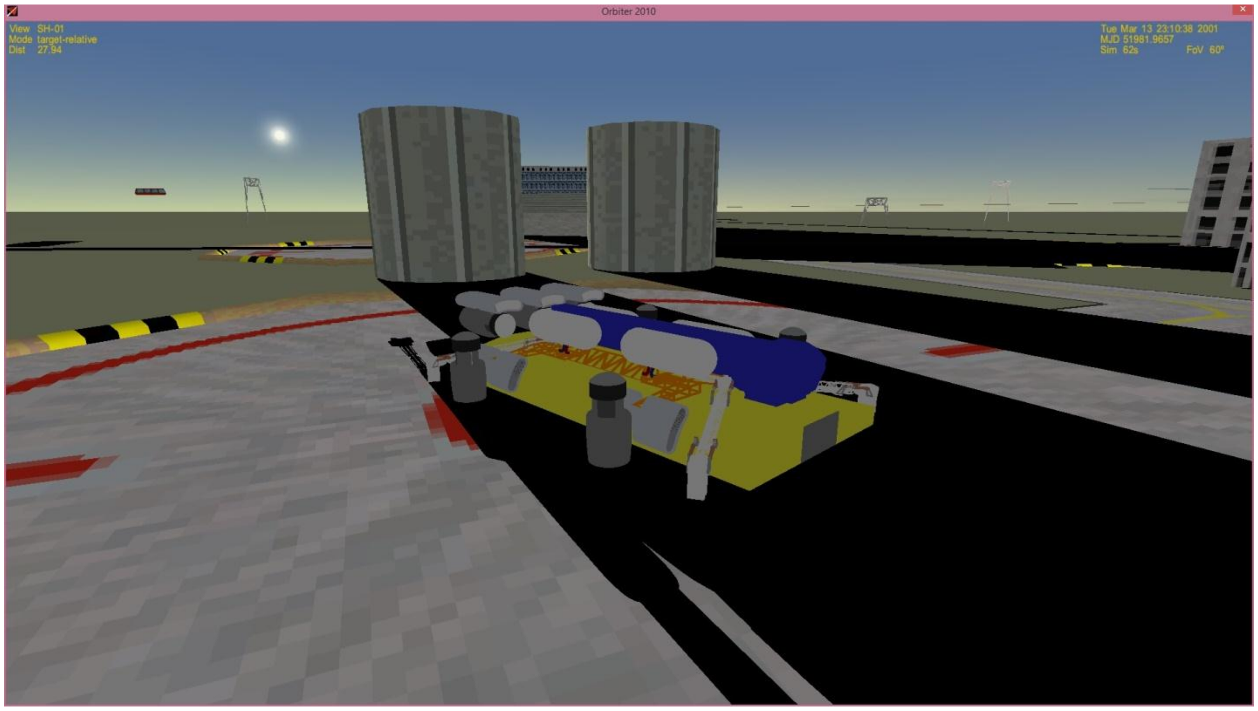


Figure 10. UAH Cargo Drone model imported from STK into Orbiter

the 108 kilometer track to a 2,000 meter track. The scenario file was edited to locate this new model at the appropriate Aitken Basin time and location on the moon. The Orbiter documentation suggests that ".dll" files are to be used to define and implement ships within the Orbiter simulator, along with ".msh" files to define the mesh and ".dds" files to define the mesh textures. As a learning exercise on how this could be done and technical tour-de-force it was decided to implement the new cargo drone model into both STK and Orbiter. The cargo drone model was thus converted from Google SketchUp to MeshLab to Orbiter. An article on the Orbiter wiki indicated that the desired development environment was Microsoft Visual Studio C++ 2008 and gave a link to a YouTube video containing the necessary instructions to work with MSVC++2008 and Orbiter to create new ship definitions. Using these steps it was possible to create the required cargo drone model and begin to work with Orbiter ship dynamically linked libraries. A visual rendering of the cargo drone in the LMD launcher is shown in **Error! Reference source not found.** and a closer view of the cargo drone in Figure 10.

It was during this time that printouts of the LMD code could finally be compared to the Orbiter API, ship modeling guide, and add-on documentation. A major struggle was the name of many of the variables used in the code, until it was discovered that several of them hadn't been translated from Russian (e.g. "BUGEL_PROC" is the percent down the track the ship cradle has progressed, as "bugel" translates to "yoke" and "proc" translates to "percent"). Once these had been properly identified, it was discovered that the Orbiter implementation did in fact not use a linear acceleration model but simply added 5% of the user’s input final speed per 5% of track length traveled. This ensures the model leaves with the user’s set final velocity at the end of the track, but does nothing to actually model the amount of speed one would receive from a track of the given length. Thus, the “linear acceleration” of the model only holds for the astronaut constrained 3G case earlier described. At the same time, early testing with OrbConnect proved highly intriguing, allowing the output of any attribute allowed in the Orbiter API to be requested over a local machine port-based connection. However, the implementation of a simple chat program in C++, as opposed to Java, to query OrbConnect proved far more difficult than initially imagined.

New C++ Federate

With two weeks to go before the event, it was almost time to begin testing and all efforts to implement a more accurate LMD federate in Orbiter capable of performing an ownership transfer were stymied. There was simply no time to develop the envisioned chat/ownership federate communicating to the more accurate LMD model in Orbiter using OrbConnect. The early decision to write the LMD ownership transfer skeleton federate in C++ meant that switching to Java to implement an OrbConnect command parser was impossible. Thus, the decision was made to scrap the Orbiter project and follow the backup plan: implement the original C++ physics model code inside the LMD ownership transfer skeleton. This would allow both sets of code to be combined and still meet the original objectives of improving the physics model used and performing an ownership transfer between the LMD and Cargo Transport federates. A further complication arose when the requested measurements of the DON visualization model used by NASA finally arrived and indicated a 202 meter track was being displayed by the DON visualization federate. The flexibility of the new code was used and the track length of the LMD federate was adjusted to reflect the track length of the DON model. This code was then used at the SEE2014 event. This did mean that the launch sequence was now only three seconds in length. Another issue was encountered where the ownership transfer would be requested but never successfully reported back to the Cargo Transport federate. Testing revealed this to occur because the launch was originally being initiated as soon as the federate joined the federation and occurring faster than the RTI could implement commands. Adding a launch delay and later on a launch command fixed the issue. Unfortunately, during the actual demonstration, lag in the VSEE connection being used meant that the entire launch sequence was not displayed. Luckily, the output windows available on both federates did indicate a successful launch, with accurate speed and error-free ownership transfer between our LMD federate written in C++ and Cargo Transport federate written in Java.

Ownership Management

A cargo drone attribute ownership for position and velocity are transferred from the CargoTrans federate to the LM federate. Using the physics model developed by the team, the LMD launches the drone and updates the drone's position and velocity attributes. When the end of the track is reached and/or the adequate ΔV is reached, the drone is considered launched. Drone's attribute ownership is returned to the CargoTrans federate where it continues to update drone's position and velocity.

The LMD joins the SEE federation creates a physical entity object called *UAHCargoDrone1*. The LMD federate subscribes and publishes to any Object with entity name *Physical Entity* in the SEE 2014 federation. Also, the federate has a call invoked into the RTI that discovers all objects in the federation with name *UAHCargoDrone1* that has an entity name *Physical Entity*.

The sequence of joining the federation between the LMD and the Skelton federate is very important for the success of this process. The LMD has to join the federation before the skeleton to ensure the ownership transfer is done properly. However, it can be improved and eliminate this contain for the future work with this federate.

Three cargo drones are created by the Skelton federate when it joins the federation. The instance of the drone objects are named *UAHCargoDrone1*, *UAHCargoDrone2* and *UAHCargoDrone3*. All attribute ownership of *UAHCargoDrone1* object instance is immediately transferred to the LMD federate. The name of the cargo drone on the LMD federate has to match the name of the drone on the skeleton, otherwise not ownership is carried on.

After ownership transfer of *UAHCargoDrone1*, the LMD publishes its position and velocity attribute data. We implemented a Launch-On-Command (keystroke) to allow us launching the LMD at any time specified during the event. A thread was implemented that accepts a keyboard stroke to change the state of the launch flag from false to true.

As soon as the user issues the drone command, the LMD federates starts simulating the movement of the done on the track. The movement of the drone on the rail is governed by physics model embedded in the federate code. Position and velocity attributes are published by the LMD federate until the end of track is reached and the ownership of the attributes is transferred skeleton federate.

Recommendations

Having now returned from the event, it would still be strongly encouraged to pursue an HLA interface utilizing OrbConnect and Orbiter. Using Java would greatly simplify a cross-platform capability to connect with a remote Orbiter session and utilize a command parser to output the requested data from any named object loaded in the scenario file. The implications of doing so are enormous, allowing Orbiter to be used both as a visualization engine and physics modeler. This would also fully divorce the HLA development code from any Orbiter add-on code future teams might decide to develop. At the very least, none of the equations or federates implemented this year had any sort of complexity that couldn't be programmed just as easily in Java. This alone would greatly simplify getting the HLA part to work. As a final note, the complaints about the Orbiter simulation being "jerky" were traced to the 2013 code being run time-constrained per federation agreement. In order to implement this requirement, the code pauses Orbiter until each time grant, advances a frame, outputs the calculated data, and then pauses again. Since the federation is time-constrained to update at most once per second, this results in a maximum of one frame per second being displayed. OrbConnect also allows this type of pause functionality to be used. Thus, any work featuring Orbiter running less than real-time can be expected to have these artifacts moving forward.

Make the LMD its own Object within the LMD federate. The federate not advancing time was linked to the ownership transfer of *UAHCargoDrone1*. A workaround was reached to make it work during the event but an optimal solution can developed.

Launch-On-Command allows only the launch of one drone during the event. After the drone is launched, the LMD federate will only be reflecting attributes of object instances in the federation. The only way to have another done launch is restart both federations which can be odd during the event. A suggestion for next year's team is to have the Skelton federate to create more launch-able drone instances (around 5 drones or more) and pass their attributes ownership to the LMD federate. Hence, the user will have more than one drone available for launch.

CARGO TRANSPORT FEDERATE

The first task in designing the Cargo Transport federate was determining what its mission would look like, and whether the design would meet the mission requirements. A simulated cargo drone must be "flown" through its mission profile to determine whether the craft could hold sufficient fuel to launch from the moon, achieve lunar orbit, transport itself to L2, dock with the station and transfer fuel, return to the moon, and land. This process illuminated a number of design considerations. For example, direct Hohmann transfers from low lunar orbit to the L2 required ideal ΔV's of roughly 4 km/s, which was found to cause the design to be very nearly impossible to close. To achieve direct Hohmann transfer from lunar orbit to L2, the Lunar Mass Driver was required to provide the cargo drone over 50% of the required launch velocity to achieve stable lunar orbit. The design team later discovered that implementing the highly complex Weak-Stability Boundary (WSB) orbital transfer method could allow the craft to transfer from lunar orbit to L2 for only 0.65 km/s of ΔV-ideal.

The problem with either of the two transition methodologies was the time required. The Hohmann, the faster of the two, required several weeks to complete, while the WSB method required far longer potentially up to several months. In either case, direct simulation of the drone transferring would not be of interest to simulate in a real time event that would only span an hour. Therefore, the team decided that only particular portions of the mission that could be demonstrated in the time allotted would be developed. The mission phases selected for demonstration were launch from the lunar surface and orbit insertion, docking and fuel transfer with the L2 station, and orbital descent and landing on the moon. Each of these separate modes was implemented as a separate Finite State Machine, and will be described in the following sections.

Launch Mode

The Cargo Drone Launch Mode contains several states that should execute sequentially. A diagram of these states is shown in Figure 11. As previously mentioned, the team found that the LMD was particularly beneficial in increasing the amount of fuel that could be transported to the L2 station. The most straightforward way to implement the LMD would have been its creation as an object within the Cargo Transport federate. However, this would prevent the reuse of the LMD federate code from the previous year's demonstration. Instead, the team elected to utilize the Ownership Management service of the HLA. The Cargo Drone object created by the Cargo Transport federate will transfer its position and velocity attributes to the LMD federate. The LMD federate would then use its internal physics model to move the drone to the end of its ramp, and then return control of the attributes to the Cargo Transport Federate. This is an important aspect, as it means that the Ownership management service can be used for physics model composability.

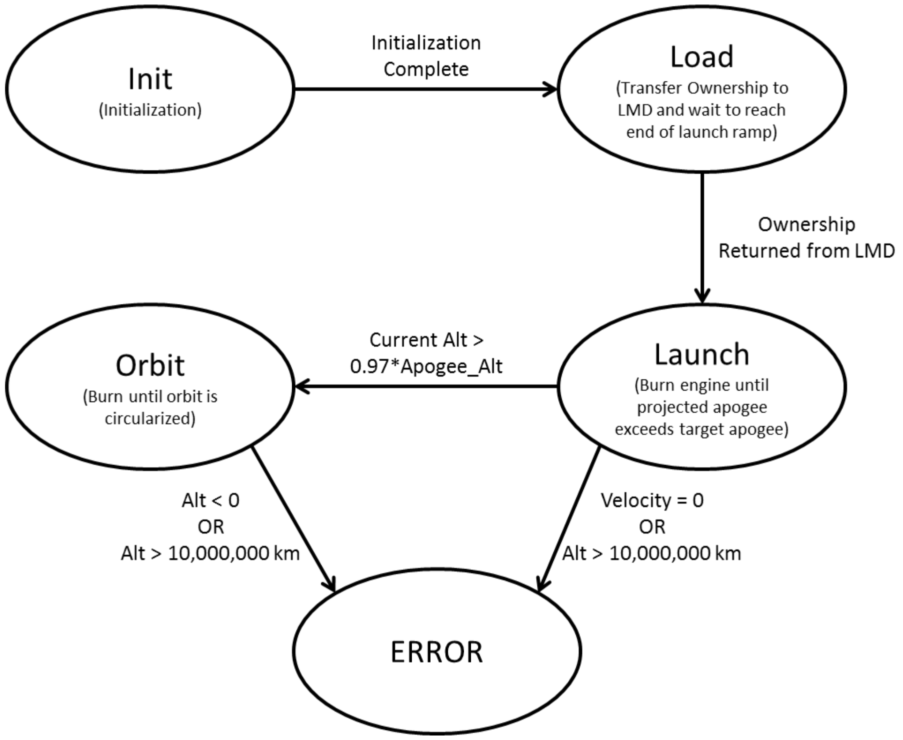

Init
(Initialization)
Initialization Complete
Load
(Transfer Ownership to LMD and wait to reach end of launch ramp)
Ownership Returned from LMD
Launch
(Burn engine until projected apogee exceeds target apogee)
Current Alt > 0.97*Apogee_Alt
Orbit
(Burn until orbit is circularized)
Alt < 0 OR Alt > 10,000,000 km
Velocity = 0 OR Alt > 10,000,000 km
ERROR

Figure 11. State diagram of the cargo drone launch

The original design of the Cargo Transport Federate had the LMD federate launching the drone as quickly as it was given control. However, implementing the code in this fashion exposed one of the potential issues with using Ownership Management: the time required for the RTI to accept the transfer. If the drone was launched immediately, the RTI had issues in providing callbacks to the federates in the order expected. To mitigate this issue, the LMD was implemented in a way that it would achieve ownership of the drone, then wait several time cycles while waiting on a user input to launch the drone. This ensures that the RTI has enough time to process the ownership transfer.

Like each of the Cargo drone states, the Launch Mode begins with the Init state. In initialization, the cargo drone is initially set at zero altitude and velocity on the lunar surface. The attitude of the drone is set to a null quaternion ([1,0,0,0]). In future builds, the simulation could be improved by initializing the quaternion to a value that would render the drone as upright. Before exiting the simulation loop, the drone advances to the Load state.

In the Load state, the cargo drone transfers ownership of the position and velocity states to the LMD federate, then waits for the drone to be launched and control returned. If control has been returned, the drone transitions to the Launch state.

In the Launch state, the drone accelerates at the maximum capability of the engine, initially equal to the drone’s weight. The drone works to maintain a zero degree flight path angle while increasing the horizontal speed. The drone then calculates the apogee altitude of the projected orbit. Once the projected apogee exceeds the desired orbital target, the engine is shut down. The drone then checks its position to determine if apogee has been reached; if so, then the drone transitions to the Orbit state.

The Orbit state restarts the drone's engine, burning to circularize the orbit. Once the perigee of the orbit is greater than or equal to the apogee, the engine is again shut down. The drone continues to propagate its state for the remainder of the duration.

In addition to the mission states described above, and additional ERROR state was implemented. The ERROR state is triggered only in the event that the drone is either greatly exceeding a feasible altitude or dips below zero. The ERROR state is intended to demonstrate to the user that a failure has occurred in the federate.

Docking Mode

Like the Launch mode, the Docking Mode contains a number of states that should execute in a sequential fashion. A diagram of the expected behavior is shown below.

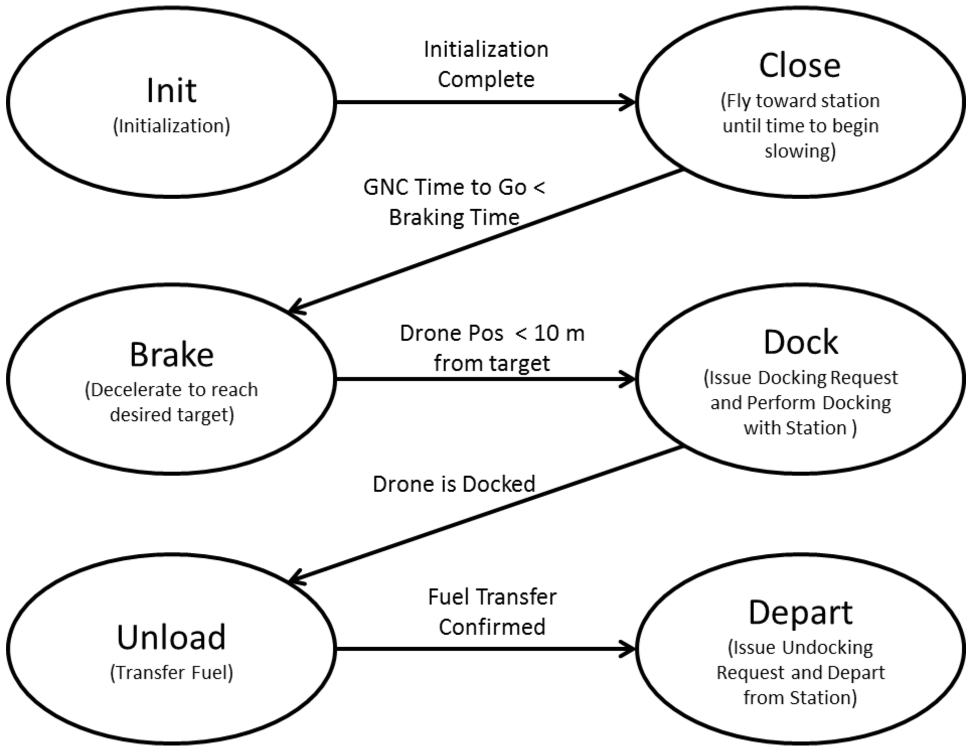

Init
(Initialization)
Initialization Complete
Close
(Fly toward station until time to begin slowing)
GNC Time to Go < Braking Time
Brake
(Decelerate to reach desired target)
Drone Pos < 10 m from target
Dock
(Issue Docking Request and Perform Docking with Station)
Drone is Docked
Unload
(Transfer Fuel)
Fuel Transfer Confirmed
Depart
(Issue Undocking Request and Depart from Station)

Figure 12. State diagram of the cargo drone docking

In the initialization state of the Docking Mode, the Cargo Drone utilizes a handful of hardcoded parameters to determine the initial position and velocity. Because the coordinate system utilized while in the Docking Mode is the Earth Moon Lagrange Point rotating reference frame, gravity is assumed to be effectively zero. While not valid, this assumption introduces very little error over the course of the demonstration. Once initialized, the drone transfers to Close state.

The Close state is a simple propagation state. It allows the drone to move along with its current speed while a timer counts down. Once the counter reaches zero, the drone transitions to the Braking mode.

The Braking state has the drone calculating the desired acceleration based on the time to arrive at the station. The algorithm then uses this to determine the acceleration required to zero the velocity in the remaining time. Typically, this desired acceleration is subject to a gain of 0.6, which ensures reasonably accurate range from the station once the drone comes to a standstill. Once the drone is within 10 meters of its desired position, the drone transitions to the Docking state.

The Docking state has the drone issuing a docking request interaction to the L2 station. Once this interaction is processed, a docking confirmation interaction will be sent to the drone. The docking confirmation sets a Boolean within the Cargo Drone object that will trigger transition to the Unload state. In the event that no confirmation is returned, a countdown timer continuously decrements and will trigger an automatic state transitions upon reaching zero.

The Unload state causes the drone to send a resource transfer scalar interaction to the L2 station. The drone then augments its mass by the amount of fuel transferred to the station. The resource transfer confirmation interaction to the drone will set another Boolean flag that will trigger transition to the Departure state. Like the Docking state, the Unload state contains a countdown timer that will automatically trigger a state transition if no callback is received within the allotted time.

The Departure state causes the drone to issue an undocking request to the L2 station. Upon receipt of an undocking confirmation the drone accelerates away from the station for the remainder of the scenario.

The Landing Mode

The Landing Mode simulates the Cargo Drone descending under power from orbit to the Aitken Basis lunar base. The flowchart for the Landing Mode finite state machine is shown below:

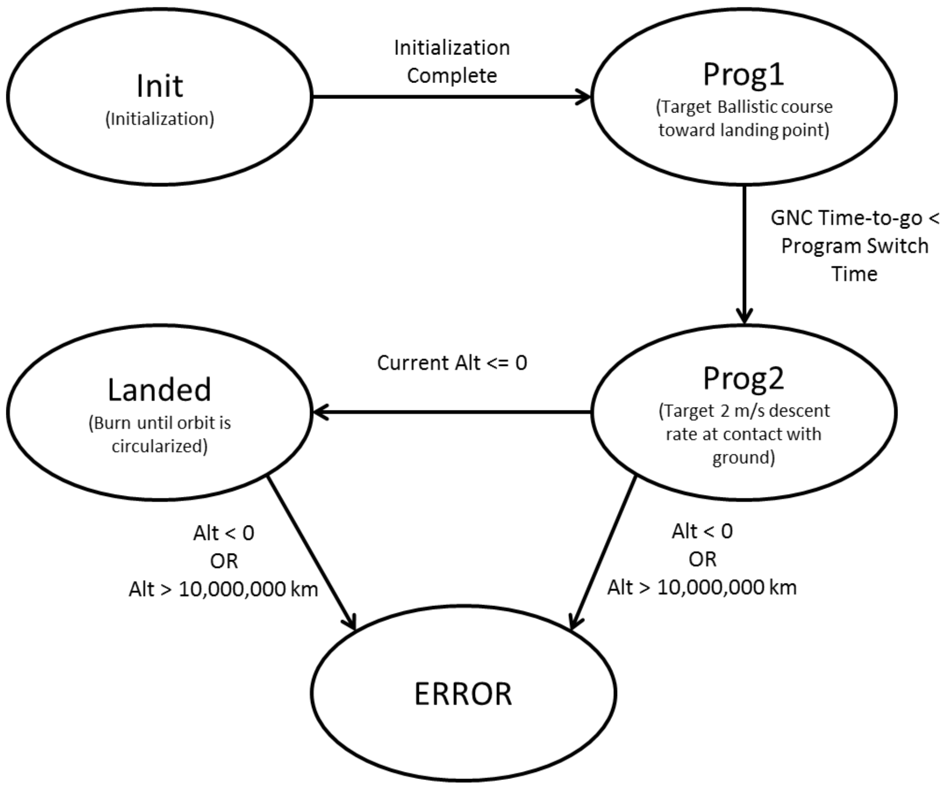

Init
(Initialization)
Initialization Complete
Prog1
(Target Ballistic course toward landing point)
GNC Time-to-go < Program Switch Time
Prog2
(Target 2 m/s descent rate at contact with ground)
Current Alt <= 0
Landed
(Burn until orbit is circularized)
Alt < 0 OR Alt > 10,000,000 km
Alt < 0 OR Alt > 10,000,000 km
ERROR

Figure 13. State diagram for landing of the cargo drone

Like all of the other modes, the Landing Mode begins in an initialization state. The drone is created in a circular orbit whose path will fly over the lunar base. The ground range from the initial state is set to 480 km; and altitude of 100 km. The drone then transitions to the Program 1 state.

The Program 1 state works to put the drone on a ballistic trajectory to intercept the base at the desired time. It does this by solving the kinematic equations to determine the acceleration required to place it on this path based on the gravity at the drone's current position. This solution is then multiplied by the program gain (typically 3) and then limited by the maximum acceleration capacity of the vehicle from current mass. This algorithm results in the drone performing a majority of the deceleration and positioning work early in the mission, which allows Program 2 to softly set the drone down on the surface. While in the Program 1 state the guidance time remaining is a simple counter that decrements based on the simulation logical time. Once the time remaining reaches the program switch point, the drone transitions to the Prog2 state.

The Program 2 state has the drone target a velocity state of 2 m/s toward the ground. It does this by solving for the constant acceleration required to reach the velocity target when the remaining time reaches zero with the gravity at the current point treated as constant. Program 2 begins active calculation of time-to-go based on the rate at which altitude is decreasing. Because of the way that this algorithm behaves, the Program 2 gain must be set to a value less than 1. The typical value for the gain is 0.6. Once the altitude of the drone becomes less than or equal to zero, the drone transitions to the landed state.

The Landed state is simply a state which ceases to propagate the Drone's motion. The altitude is set to zero at the last known position, and the velocity is zeroed. This state simply publishes this position and velocity for the duration of the scenario.

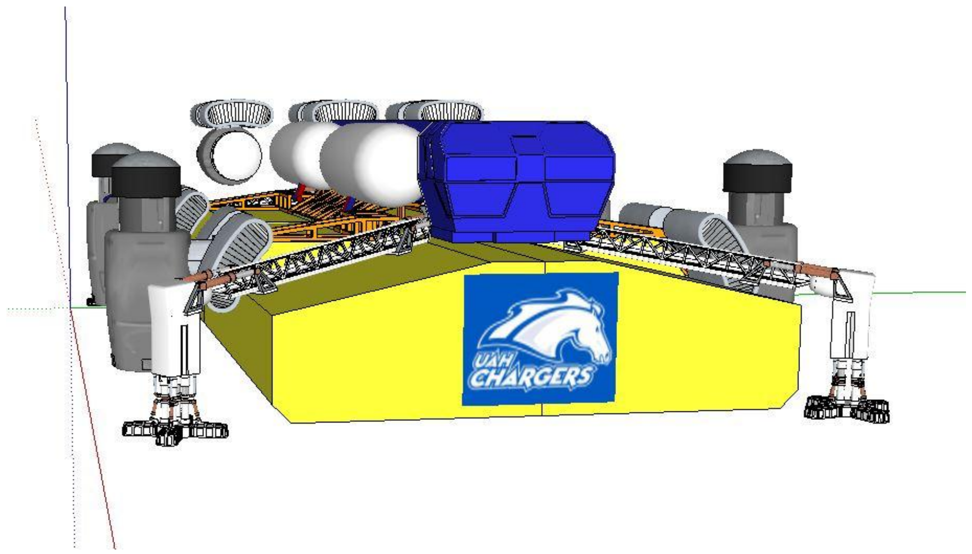


Figure 14. Three dimensional model used for visualization of the cargo drone

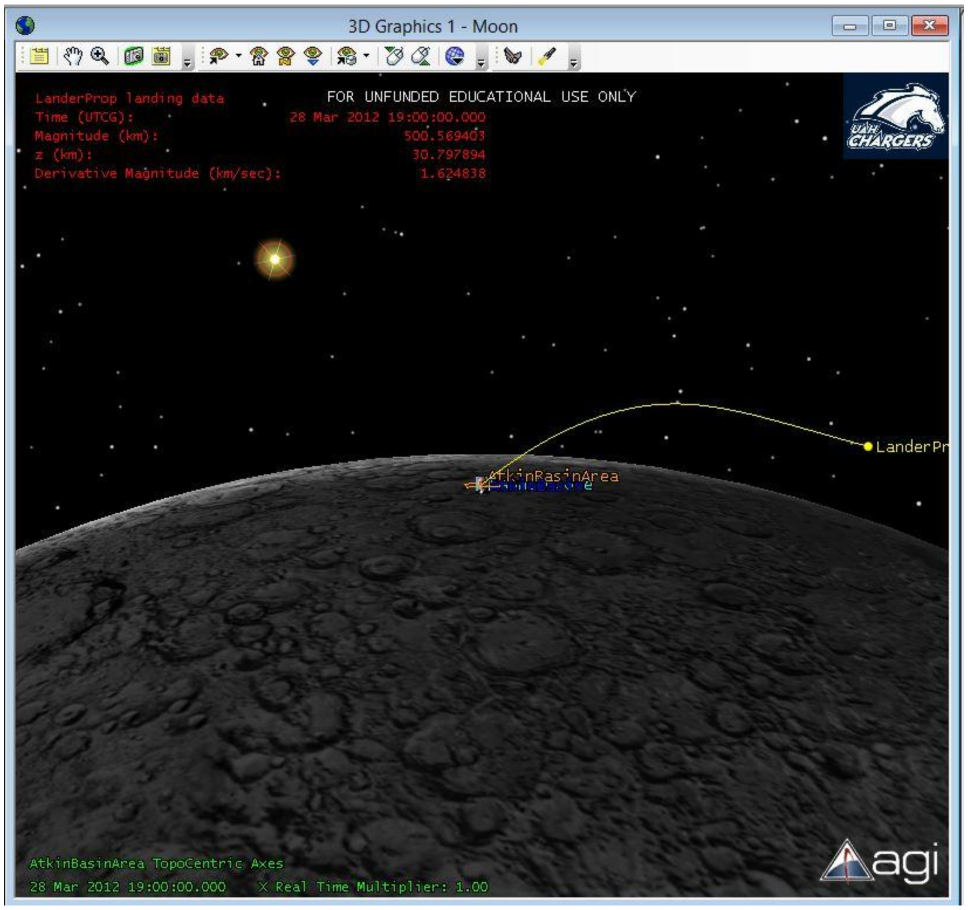
The 3D model used to visualize the cargo drone was created using Sketchup. The base model was downloaded from the 3D warehouse then edited in Sketchup to give it that UAH feel. The model was then exported to a ".dae" file to be used in STK. The model was later exported as a ".obj" file and reduced in blender so that it could be used in the DON Tool also.

STK 10

The commercial software program STK 10 was used to visualize the landing of the Cargo Drone on to the moon after it had returned from L2. The team decided to use STK as a way to visualize the cargo drone because the DON tool would not be ready until the week of the event. We needed a way to visualize the results that were being generated by our propagator so that we could make sure the propagator was generating the correct data.

2014 Implementation:

Even though the TCP/IP client was not finished we were still able to use STK to test our propagator and to showcase the landing of our Drone at the competition. We did this by outputting the position, velocity and attitude (quaternions) from the propagator into a text file. This data was then used to create .e (Position file in STK) and .a (attitude file in STK) files. These files can be loaded into STK to display the movement of the drone. The first step in STK is to create a scenario. Once you create a scenario you can add objects like satellites, buildings, ground vehicles, targets and much more. To create our landing scenario we used the Satellite, Facility, and Area Target Objects. The following paragraphs will discuss the creation of the objects that were used in the Landing Scenerio: the Lander, Launcher, LMD, LaunchDrone, AtkinBasinArea, and AtkinBasin Target.

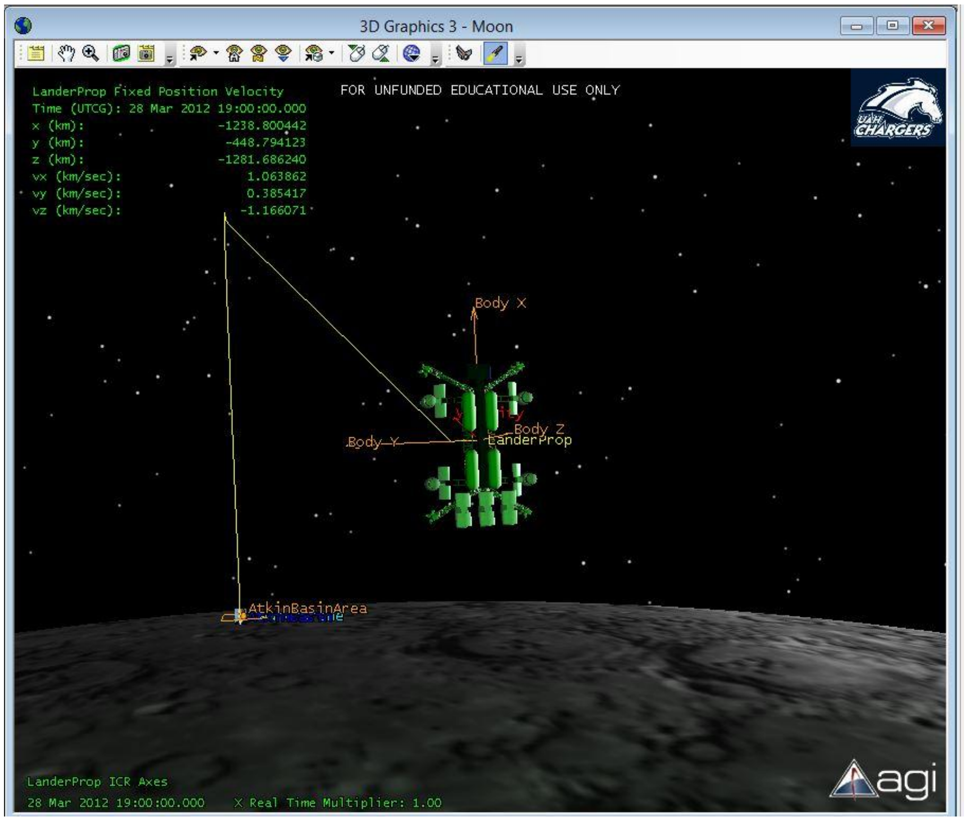


```
LanderProp landing data
Time (UTCG):                        28 Mar 2012 19:00:00.000
Magnitude (km):                                 500.569403
z (km):                                          30.797894
Derivative Magnitude (km/sec):                    1.624838
```

```
AtkinBasinArea TopoCentric Axes
28 Mar 2012 19:00:00.000     X Real Time Multiplier: 1.00
```

Figure 15. Visualization of cargo drone landing

Lander:



```
LanderProp Fixed Position Velocity
Time (UTCG): 28 Mar 2012 19:00:00.000
x (km):                -1238.800442
y (km):                 -448.794123
z (km):                -1281.686240
vx (km/sec):               1.063862
vy (km/sec):               0.385417
vz (km/sec):              -1.166071
```

```
LanderProp ICR Axes
28 Mar 2012 19:00:00.000    X Real Time Multiplier: 1.00
```

Figure 16. Visualization of

The satellite is the generic space vehicle in STK. To make the satellite act like a cargo drone 3 primary settings were changed. First the default 3D model of the satellite was replaced with the model of the cargo drone that we created using Google Sketch-up. Next the orbit was replaced with the Ephemeris (.e) file that contained position and velocity of the drone generated from the propagator. And lastly the Attitude was replaced by the .a file which contained the quaternion data generated from the propagator. Once these two files were uploaded STK automatically draws the trajectory or the path that the lander will travel. This was very useful because we could easily see that our propagator was working correctly. Another advantage from using STK for testing is that it is not constrained to real time. You can adjust the simulation to run at whatever speed you want. The cargo drone takes around 12 minutes to land in real time, but since we were using STK we could run the whole landing routine in just a few minutes. This cut the amount of time spent testing the quaternions generated from the propagator significantly. It was easy to ensure that the drone was tilting and rotating correctly when running the simulation at 8 or 16 times speed. STK also has the ability to display real time data on the screen. We displayed the drone's distance from the landing site, height from the surface of the moon, the magnitude of the drone's velocity and the x,y, and z position and velocity values in relation to the center of the moon. As seen in the red and green text in the picture below.

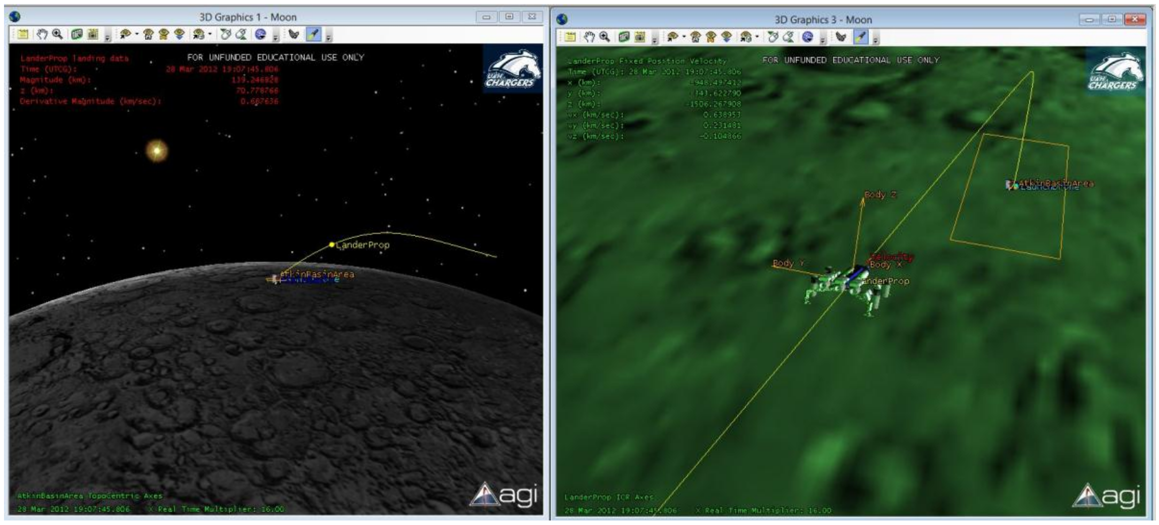


Figure 17 Visualization of

Launcher

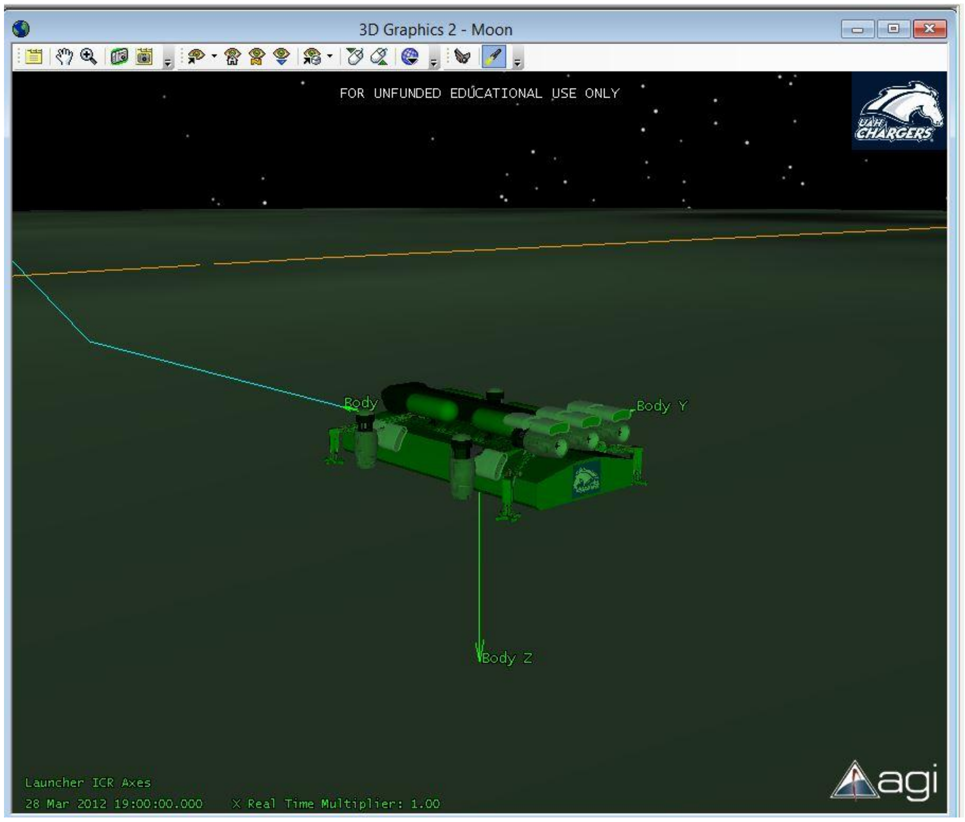

3D Graphics 2 - Moon
FOR UNFUNDED EDUCATIONAL USE ONLY
UAH CHARGERS
Body
Body Y
Body Z
Launcher ICR Axes
28 Mar 2012 19:00:00.000 X Real Time Multiplier: 1.00
agi

Figure 18. Visualization of

The intent of the Launcher model was to show the drone being launched from the LMD and then transition into moon orbit. We were able to add the launching data that was generated from the LMD federate but the orbiting data was not added. This will not be an issue once the TCP/IP client is implemented since STK will display whatever data the cargo drone federate is publishing or subscribing to. As you can see above, we used two different models to visualize the cargo drone. The Launcher drone includes a yellow cargo tank where the fuel or other supplies that are being delivered to L2 are stored. The Lander drone does not have the tank, because it was dropped off at the station.

Surface Objects

A few other objects were added to the scenario to showcase the overall the LMD and Atkin Basin landscape. The coordinates of Atkin Basin from the environment FOM were input into the area target STK object to display the outline of the Area where the LMD and all the other federate objects would located (the orange outline in the figure below.

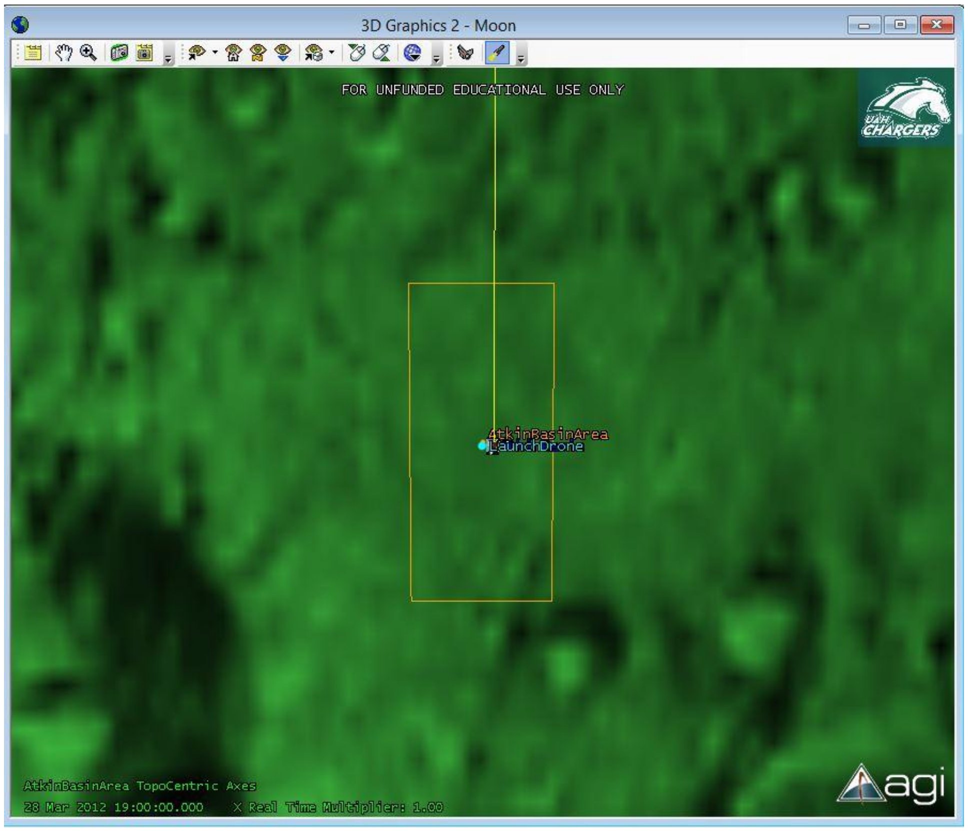

3D Graphics 2 - Moon
FOR UNFUNDED EDUCATIONAL USE ONLY
UAH CHARGERS
AtkinBasinArea
LaunchDrone
AtkinBasinArea TopoCentric Axes
28 Mar 2012 19:00:00.000 X Real Time Multiplier: 1.00
agi

Figure 19. Visualization of

The LMD model that was created in Orbiter 2010 was input into an STK Facility object to show the LMD's location on the moon. The cargo drone model was also created as a Facility to show the drone on the LMD launch pad. See Figures Below.

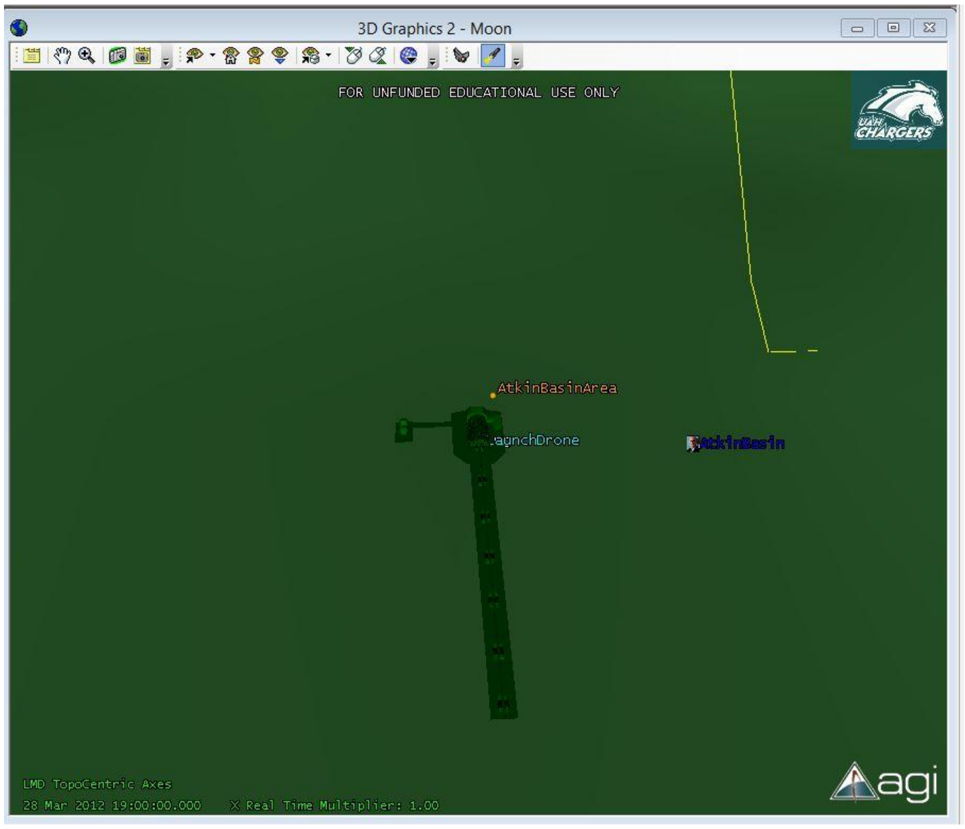


Figure 20. Mass driver visualization

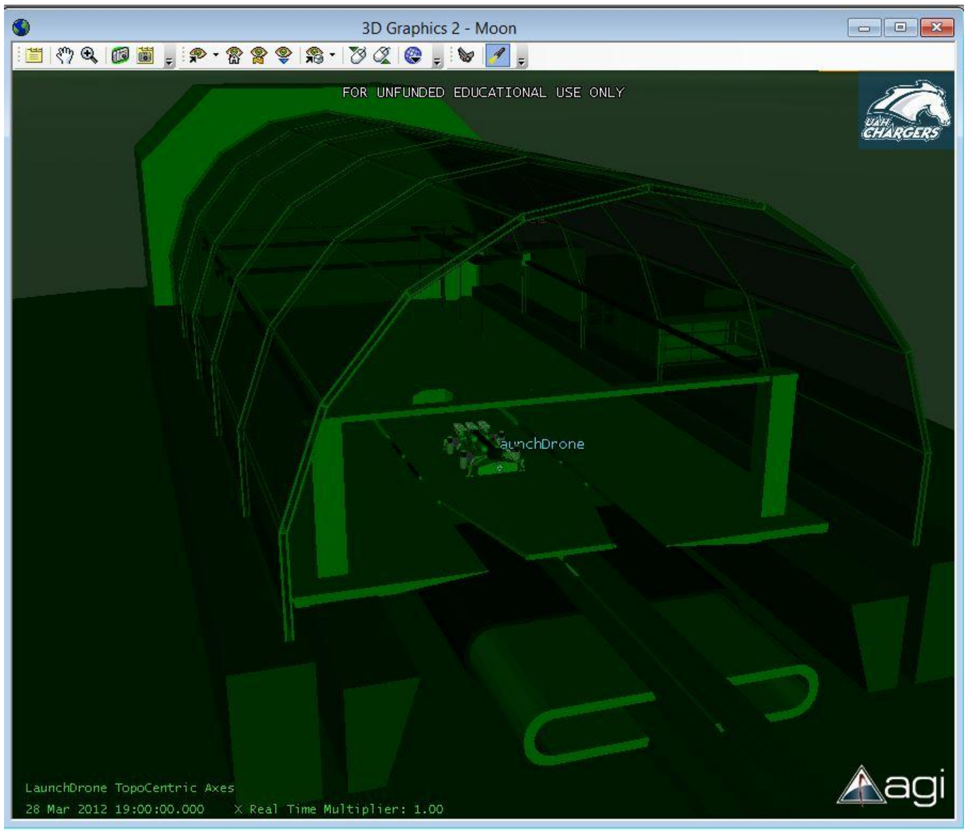

3D Graphics 2 - Moon
FOR UNFUNDED EDUCATIONAL USE ONLY
UAH CHARGERS
LaunchDrone
LaunchDrone TopoCentric Axes
28 Mar 2012 19:00:00.000 X Real Time Multiplier: 1.00
agi

Figure 21. Visualization of

Future Implementation Advice:

The original plan was to create a TCP/IP client in the Java cargo drone federate that would send position and attitude data that was generated from the cargo drone propagator to STK in real time. This project was cut short so that higher priority tasks could be completed in time for the event. It is *highly recommended* that the next team implements the TCP/IP client in next year's code. If used then the LComSat, LMD and CargoTrans federates and any other federates connected to the network can be seen on one screen. Also, it is the opinion of the team that STK looks better and performs better than the DON tool that was used during the demonstration. This also gives the team the added benefit of showing what they want to show on the screen instead of having NASA in control where your federates may or may not be shown at the correct times.

Tips when using STK:

Make sure your using the correct reference frame. Our data was based off of moon fixed.

Be prepared for terrain issues. STK uses slightly different moon terrain data than the DON tool uses. Our propagator was based off of the DON tool. There was around a 15 km difference between the two tools which caused the landing drone's final state to appear to be hovering above the surface. We were able to add some data points to the .e file to get the lander to actually reach the surface on STK, but this will not work when streaming data directly through the network.

Make sure your 3D models are 15,000kbs or less. The original Cargo Drone model was 8MB. The DON tool could not handle that size. The file size was reduced to 13,000kb using blender.

Scale the size of the drone model. Our model ended up being almost half the size of the L2 outpost. This is not very realistic but it made the drone clearly visible during the presentation.

Obtain a full license for STK 10. Dr. Petty has been given an AGI representative's contact information who can provide a license.

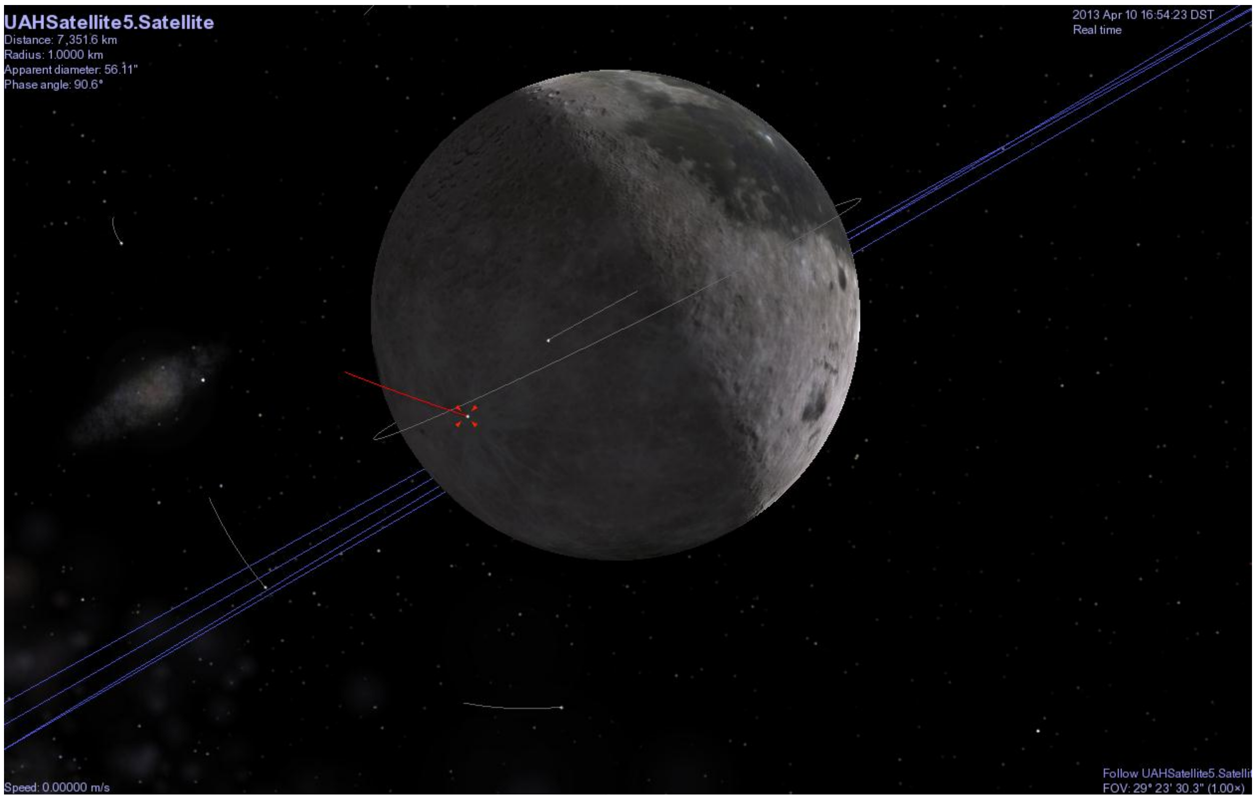


Figure 22. LComSat satellite constellation as seen from the Munich University developed L2 Outpost and visualized with their Celestia software

DISCUSSION

CONCLUSIONS

The creation of the CargoTrans federate added needed infrastructure and cohesion to the lunar colony. By connecting the production of LO_2 on the lunar surface with those entities that had a need for it, the CargoTrans federate illustrates viability and purpose for the lunar colony. Likewise the simulation of the federate through the HLA infrastructure illustrates the handshaking, negotiation, and interoperability needed for such an endeavor. The generic nature of the HLA standard requires the definition and negotiation of interactions between federates as do all interfaces within a developmental world. The multiple ownership transfers of the cargo drones provided just such an example. In a similar manner the LComSat federate provided cohesion through the interconnection of all entities regardless of location.

Working through the forum paradigm allowed all teams to gain knowledge when any one team had an issue. The tech forum and quick response of the NASA representatives provided a much needed conduit for working through software and interconnect issues, as well as, create consistency in the overall federation plan. [1] [2] [3] [4] [5]

REFERENCES

[1] IEEE, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) –Object Model Template (OMT) Specification,* Institute for Electrical and Electronic Engineers, 2010.

[2] IEEE, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) –Federate Interface Specification,* Institute for Electrical and Electronic Engineers, 2010.

[3] IEEE, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) –Framework and Rules,* Institute for Electrical and Electronic Engineers, 2010.

[4] Moller, Bjorn, "The HLA Tutorial: A Practical Guide for Developing Distributed Simulations," Pitch Technologies, Linkoping, Sweden, 2012.

[5] O'Neil, Daniel, "SEE 2014 Federation Agreement," Simulation Exploration Experience, NASA, Marshall Space Flight Center, 2014.