A Hardware-In-The-Loop Simulator for Software Development for a Mars Airplane

Stefan E. Slagowski [1] and Justin E. Vican[2]
The Charles Stark Draper Laboratory Inc., Cambridge, MA, 02139

and

P. Sean Kenney[3]
NASA Langley Research Center, Hampton, VA, 23681

Draper Laboratory recently developed a Hardware-In-The-Loop Simulator (HILSIM) to provide a simulation of the Aerial Regional-scale Environmental Survey (ARES) airplane executing a mission in the Martian environment. The HILSIM was used to support risk mitigation activities under the Planetary Airplane Risk Reduction (PARR) program. PARR supported NASA Langley Research Center's (LaRC) ARES proposal efforts for the Mars Scout 2011 opportunity. The HILSIM software was a successful integration of two simulation frameworks, Draper's CSIM and NASA LaRC's Langley Standard Real-Time Simulation in C++ (LaSRS++).

I. Introduction

NASA's Mars Scout Opportunity was created to enlist proposals for innovative investigations that complement NASA's core Mars Exploration Program. NASA Langley Research Center (LaRC) teamed with the Jet Propulsion Laboratory, NASA Goddard Research Center, Lockheed Martin Astronautics, Aurora Flight Sciences, The Charles Stark Draper Laboratory, Malin Space Science Systems, and several prominent academic researchers to participate in the opportunity. The team established science goals for the project that required a regional survey of Mars. Trade studies determined that an aircraft provided the best opportunity to complete the science objects of the team. The team proposed a project where an aircraft would be released into the atmosphere of Mars to perform an aerial survey of the Martian surface[1-4]. This mission was named the Aerial Regional-scale Environmental Survey (ARES) of Mars and was proposed for both Mars Scout 2007 and 2011. Unfortunately, ARES was not selected for either opportunity. However, ARES demonstrated that airplanes are a feasible approach to planetary exploration.

The reference mission for the airplane was to fly a pre-determined, highly accurate pattern of relative paths at a fixed height above ground level to accomplish the science objectives. The reference suite of science payload sensors included a mass spectrometer to sample atmospheric gases along the flight path; cameras and a spectrometer to catalog the surface composition and context in high resolution; and a magnetometer to determine sub-surface structure. Targeted for the highly desirable southern highlands of Mars that are currently inaccessible to rovers because of the high altitude, the nominal flight would have covered several hundred kilometers of terrain in over one hour. Once the flight had been completed, the airplane would have executed a controlled landing on the surface of Mars.

In support of NASA LaRC's sponsored Planetary Airplane Risk Reduction (PARR) program, Draper Laboratory developed a Hardware-In-The-Loop Simulator (HILSIM) to support flight code development on the ARES avionics. The PARR program supported the ARES Mars Scout 2011 proposal and evolved from the previous ARES Mars Scout 2007 proposal. The goal of this program was to address technical risks associated with flying an autonomous airplane on Mars. One risk was that the proposed avionics were not capable of flying the airplane and handling all the science data during the plane's flight time. This risk was addressed through demonstrations on the HILSIM. In addition, the Draper Flight Software (FSW) developers gained experience developing application code on the avionics executing the VxWorks real-time operating system.

[1] Modeling and Simulation Engineer, Modeling and Simulation Group, 555 Technology Square, MS 93
[2] Modeling and Simulation Engineer, Modeling and Simulation Group, 555 Technology Square, MS 93
[3] Aerospace Engineer, Simulation Development and Analysis Branch, MS 125B , AIAA Member

The HILSIM facility consisted of a simulation workstation, a Simulation Interface Unit (SIU), and the hardware under test. The hardware under test was a Broad Reach Engineering (BRE) provided Integrated Avionics Unit (IAU) which consisted of a compact PCI backplane, a Rad750 Central Processing Unit (CPU) board, various Input/Output (IO) boards, and power distribution boards. The simulation workstation executed the real-time, 6 Degree-of-Freedom (6-DOF) simulation of the airplane flying in the Martian environment that was previously developed by LaRC[5,6]. The simulation software was a combination of two simulation frameworks: Draper Laboratory's CSIM simulation framework and LaRC's Langley Standard Real-Time Simulation in C++ (LaSRS++) simulation framework.

II. HILSIM Architecture

The functions and capabilities of the simulation workstation, the SIU, and the IAU are detailed below. A block diagram of the HILSIM facility is shown in Figure 1 and a photo of the HILSIM in the Draper Simulation Lab is shown in Figure 2.

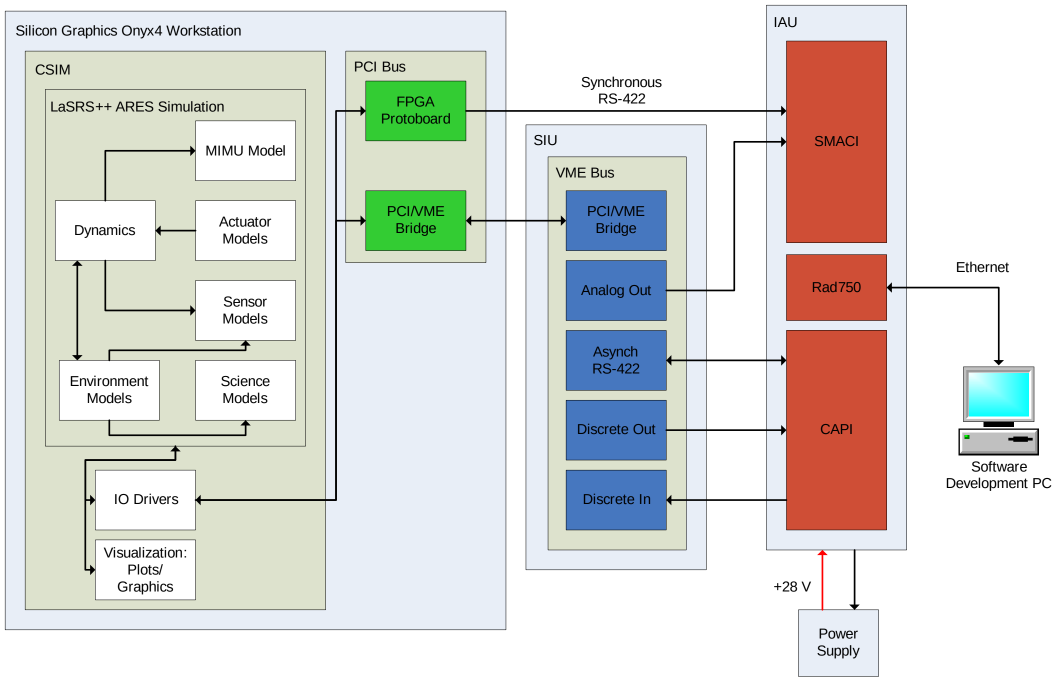

Figure 1. HILSIM Block Diagram

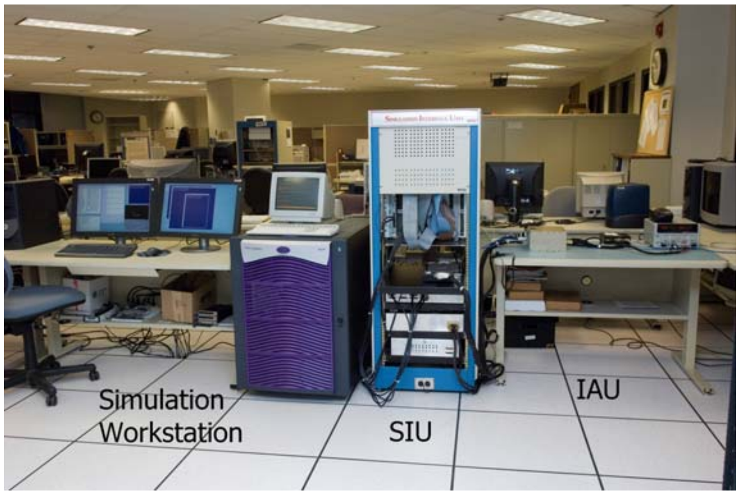

Figure 2. The HILSIM in the Draper Simulation Lab

A. Simulation Workstation

The simulation workstation used for the HILSIM was a Silicon Graphics, Inc. Onyx4 four processor workstation executing the IRIX operating system. The workstation executed the simulation software and distributed the simulation among the four processors. The simulation software consisted of the 6-DOF simulation of the airplane in the Martian environment, a user interface, IO handling, and data logging. The simulation software is outlined in more detail below.

The workstation interfaced with the IAU through the SIU using a set of VME boards. In addition, the workstation also hosted a Field Programmable Gate Array (FPGA) prototyping PCI board. The FPGA simulated the outputs of a Honeywell Miniature Inertial Measurement Unit (MIMU) directly to the MIMU interface on the IAU. This MIMU interface consisted of three RS-422 synchronous serial signals: a 1 MHz clock signal, a 200 Hz data enable signal, and a 200 Hz data signal. The simulation software controlled the operation of the FPGA and provided the simulated MIMU measurements.

The 200 Hz data enable signal was also used to schedule the execution of the simulation. Separate circuits on the FPGA prototyping board output a signal at the rising edge of the 200 Hz data enable pulse which was looped back into an external interrupt port on the SGI. This interrupt triggered the execution of the simulation. This ensured that the simulation generating the MIMU measurements and the FPGA simulating the MIMU outputs were in lockstep. This capability was added after initial results integrating the IAU into the HILSIM revealed that time between generating the truth states and when the IAU received them was drifting during the hour long simulation of the airplane's flight. This resulted in dropped data.

B. Simulation Interface Unit

The SIU consisted of a VME chassis, breakout boxes, a LED panel, and toggle switch panel housed in a standard rack. The primary function of the SIU was to provide the IO interface between the simulation workstation and the IAU. In addition, the breakout boxes provided test points where the signals between the IAU and simulation workstation were monitored.

The VME chassis housed the IO cards required to interface with the IAU. The simulation workstation had access to the VME bus through a PCI to VME bus adapter. The SIU only interfaced with the IAU through serial and discrete interfaces. Analog IO boards were also housed in the VME chassis for future expansion of the simulation to include the analog interfaces to the IAU.

The toggle switch panels were used to provide test discrete signals that were monitored by the simulation software. The simulation software could also modify the state of the LED panel in order to provide insight into the execution of the simulation. LEDs were configured to indicate when the FSW began execution and receipt of commands from the IAU. This provided a quick visual indication of the correct operation of the HILSIM.

C. Integrated Avionics Unit

The IAU is currently maintained at Draper on loan from LaRC. It is being used in research and development projects to further technologies for planetary navigation. The IAU consists of the following:

- Rad750 CPU
- Command and Payload Interface (CAPI) – The CAPI board provides RS-422 asynchronous interfaces, discrete inputs, and discrete outputs.
- State of Health Monitoring and Attitude Control Interface (SMACI) – The SMACI board provides the interface to the MIMU, analog inputs, and analog outputs.
- Camera and Storage Interface (CASI) – The CASI board provides 512 Mbytes of Dynamic RAM. The CASI purchased for PARR does not implement any external interfaces.
- Solar Array and Charge Interface/Payload and Power Interface (SACI/PAPI) – The SACI is

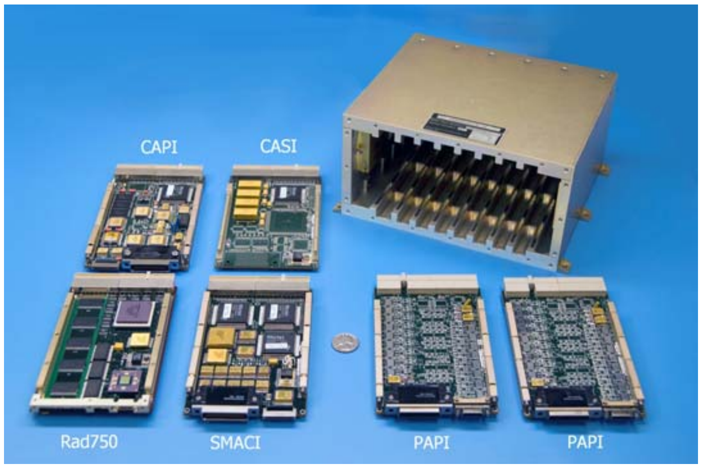

Figure 3. BRE IAU

mounted behind the IAU backplane and contains a 28 V power switch for the MIMU and a FPGA to command the power and pyrotechnic switches. The physical switches commanded by the SACI are located on the PAPI boards.
- Chassis – 8 slot cPCI backplane with 5 slots for Command and Data Handling (C&DH) boards and 3 power distribution boards.

Figure 3 shows the IAU enclosure with the front panel removed and the Rad750 CPU, CAPI, SMACI, CASI, and PAPI boards. In addition to the boards shown, a commercial Ethernet board was also housed in the IAU chassis in one of the C&DH slots to support FSW development and testing. The IAU consists of build to print boards designed for previous missions in order to reduce costs and delivery time. The functionality of the IAU met the requirements for the PARR HILSIM to provide a flight computer similar to one anticipated for use on the ARES mission. For PARR, the power and pyrotechnic interfaces were not tested closed loop in the HILSIM.

III. Simulation Software Architecture

The simulation software was composed of the Draper CSIM and the LaRC LaSRS++ simulation frameworks. The simulation software was responsible for executing a 6-DOF simulation of the airplane flying in the Martian environment, providing a user interface, and interfacing with the IAU through the SIU. The distribution of this functionality of the simulation between the two frameworks as well as their integration is discussed below.

A. CSIM

The CSIM simulation framework was developed at Draper Laboratory and has been used successfully on multiple programs including space, undersea, reentry vehicle, rotary vehicle, and guided munitions applications. CSIM is capable of executing a 6-DOF simulation of a vehicle, scheduling it for real-time execution, and performing all IO activities for Hardware-In-The-Loop (HWIL) applications. In addition, CSIM can provide a user interface for insight into and control of the simulation execution.

For the PARR HILSIM application, CSIM provided the user interface, engineering visualization, scheduling, and IO handling. A 6-DOF simulation of the ARES airplane was previously developed using LaSRS++ and it was decided at the start of the program to take advantage of the capability of LaSRS++ to be rehosted in another framework[7,8] as opposed to implementing the simulation again in CSIM. The steps performed to integrate the LaSRS++ simulation of the ARES airplane into CSIM are discussed in more detail below.

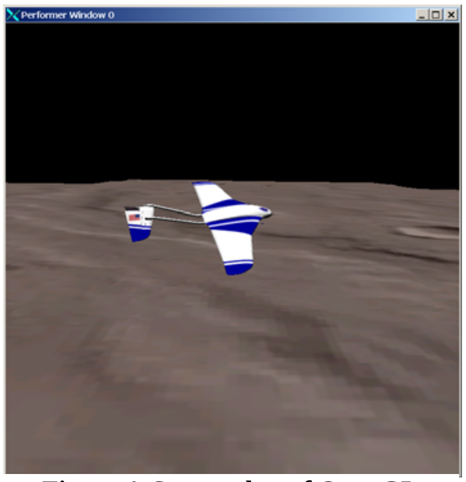

Figure 4. Screen shot of OpenGL Performer ARES visualization

The engineering visualization was implemented using OpenGL Performer. Performer is a low cost scene graph provided by SGI for creating real-time graphics applications. A three dimensional model of the ARES airplane provided by LaRC and Mars Orbiter Laser Altimeter (MOLA) data of the science target area terrain were converted to a format usable by Performer. The MOLA terrain was used for visualization purposes and a separate terrain model was implemented in the 6-DOF simulation to determine the airplane's altitude above ground level. In order to reduce jitter in the visualization, the local geographic frame at the southwestern corner of the terrain tile was translated to the Performer origin frame to reduce the magnitude of the position of the airplane relative to the origin of the scene. A screen shot of the Performer visualization is shown in Figure 4.

When the simulation software was started, two separate threads were created for executing the 6-DOF simulation and the IO handling respectively. In addition, a separate Performer process was also created to handle the rendering of the visualization. Both threads and the Performer process were assigned to separate dedicated processors. Configuring the processors in this way ensured that only the desired tasks were executed and enabled the simulation to run in real-time. Passing data between the simulation and IO threads was handled through shared memory. Passing data between the simulation and the Performer task was handled using the Performer DataPools. Performer DataPools are very similar to shared memory arenas with the added ability to lock and unlock the memory for multiprocessor applications.

The IO thread monitored the hardware interface boards for inputs from the IAU and delivered the data from the simulation to the IAU. Most communications between simulated components and the IAU were implemented using asynchronous RS-422 interfaces. These components included the control actuation surfaces, the atmospheric data system, the radar altimeter, the communications system, and the science instruments. In addition to the serial interfaces, discrete IO was also used by the IAU to control the flight of the ARES airplane. The IAU used discrete outputs to pulse the rocket motor and to simulate the firing of a pyro to cut the drogue chute. In order to synchronize the simulation and the FSW, one of the discrete outputs of the IAU was configured to signal the simulation when the FSW began execution.

Although these interfaces were lower fidelity than what would actually be flown, they provided representational IO rates to the IAU. Many of the interfaces on the actual system had yet to be defined so a set of interface control documents was derived in order to design and implement the models of the components. Realistic science data were not generated and the science data packets were populated using 32 bit counters. Use of counters eased the verification that all the science data was received by the IAU from the simulated science instruments and sent back to the simulated communications interface.

Although CSIM has the capability to log simulation variables it could not do so at the required rate to capture all the serial data. Therefore a separate data logger was implemented to capture serial data as it was received and transmitted. Memory was allocated during simulation initialization for a defined set of interfaces. All data passing through those interfaces were copied to memory. At the completion of the simulation, the logged data in memory were written to a file for later analysis. The name of the file was auto-generated to include the interface name, the date, the time, and a unique simulation run identifier.

The simulation started in an initialization mode and monitored a discrete input channel for a signal from the IAU that the FSW had begun execution. The simulation software then initiated the execution of the FPGA MIMU simulator and transferred to run mode to begin execution scheduled by the 200 Hz outputs from the FPGA.

B. LaSRS++

LaSRS++ was developed at the NASA LaRC to support activities at the Simulation Development and Analysis Branch and supports a wide range of simulation activities, ranging from batch tests to real-time pilot and hardware in the loop simulations. Current and past simulation projects include transports, general aviation, and fighter aircraft as well as the ARES Mars airplane, a planetary lander, and NASA's next generation launch vehicle. The framework includes a suite of classes that allow new vehicles to be simulated with a maximum amount of reuse of existing code.

The LaSRS++ framework is composed of six distinct components: simulation control, real-time services, the virtual environment, hardware communication, hardware interfaces, and the user interface. The virtual environment is composed all of the simulation models and the environmental models with which a simulation model might interact with. Figure 5 illustrates a high level perspective of the virtual environment.

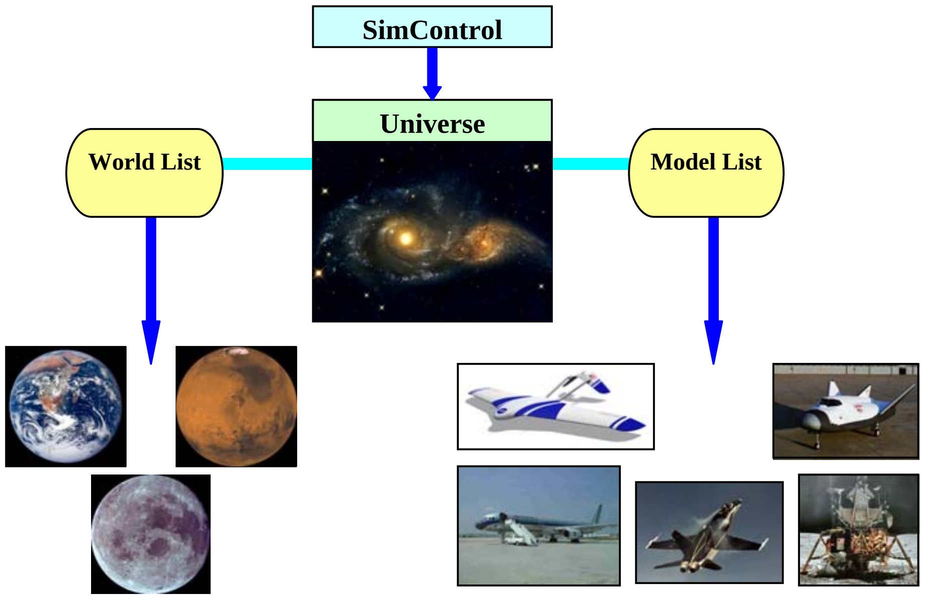

Figure 5. LaSRS++ Virtual Environment

The Universe class encapsulates the entire simulated reality. It contains a list of all the simulation models in the virtual environment and the relative geometry between them. At the highest level, a model is an object that a pilot or other models interact with. The Universe class also contains a list of Worlds that represent different environments in which the models may reside. A World is any celestial body that has a gravity model and may also have an atmospheric and navigation related models. SimControl is a class utility that provides global access to the simulation mode, time, time step, and multi-CPU synchronization symbols to the Universe and other framework classes. The Universe examines the current simulation mode and instructs the simulation models and worlds to perform tasks corresponding to that mode.

Figure 6 is a UML diagram that illustrates the hierarchy of classes used by LaSRS++ simulation models. The base class PositionalModel occupies a location within the virtual reality and can exhibit movement. A PositionalModel has a position and orientation and can dynamically update these states. A Vehicle is a PositionalModel that reacts to external forces and moments. A Vehicle therefore has accelerations and integrates its states. An Aircraft is a type of Vehicle whose external forces and moments are significantly influenced by the atmosphere. An Aircraft has additional states such as angle-of-attack and sideslip. The Vehicle and Aircraft classes are abstract and cannot be instantiated. A simulation model must derive from one of these classes to obtain the base class behaviors and attributes. In this illustration, the F18a class derives from the class Aircraft.

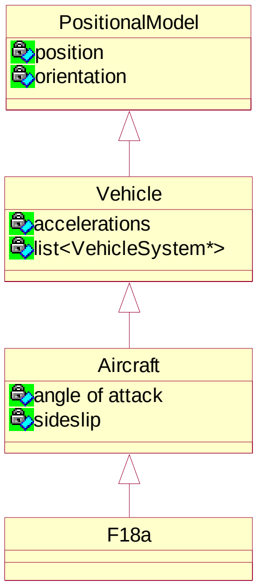

Figure 6. Simulation Model Hierarchy

All simulation models developed with LaSRS++ derive from either the Aircraft class or the Vehicle class. Figure 7 provides a simplified representation of the architecture employed by LaSRS++ simulation models. A simulation model is composed of three major parts:

- Subsystem models descended from SimulationModel. Examples of subsystem models are aerodynamic, propulsion, and control system models.
- Mediator classes descended from VehicleSystem. VehicleSystems follow the mediator design pattern to decouple subsystem models from other parts of the aircraft. VehicleSystems construct the subsystem model and handle IO for the subsystem model. In Figure 7 the F18aAeroSystem descends from VehicleSystem. It creates the F18aAero object and it retrieves from various sources the inputs to the F18aAero object and instructs it to perform its computations. Finally, the F18aAeroSystem makes the F18aAero model’s outputs available to other VehicleSystems.
- The simulation model of interest descends from Vehicle. In this example, the F18a model descends from Vehicle and instantiates an F18aAeroSystem model and places this on the list of VehicleSystems. The F18a represents an F-18A aircraft by creating all of the systems that define the aircraft’s behavior and placing them on the list of VehicleSystems as well. While not shown in this diagram, the F18a also creates an F18aPropulsionSystem, F18aContolSystem, F18aLandingGearSystem, and so on.

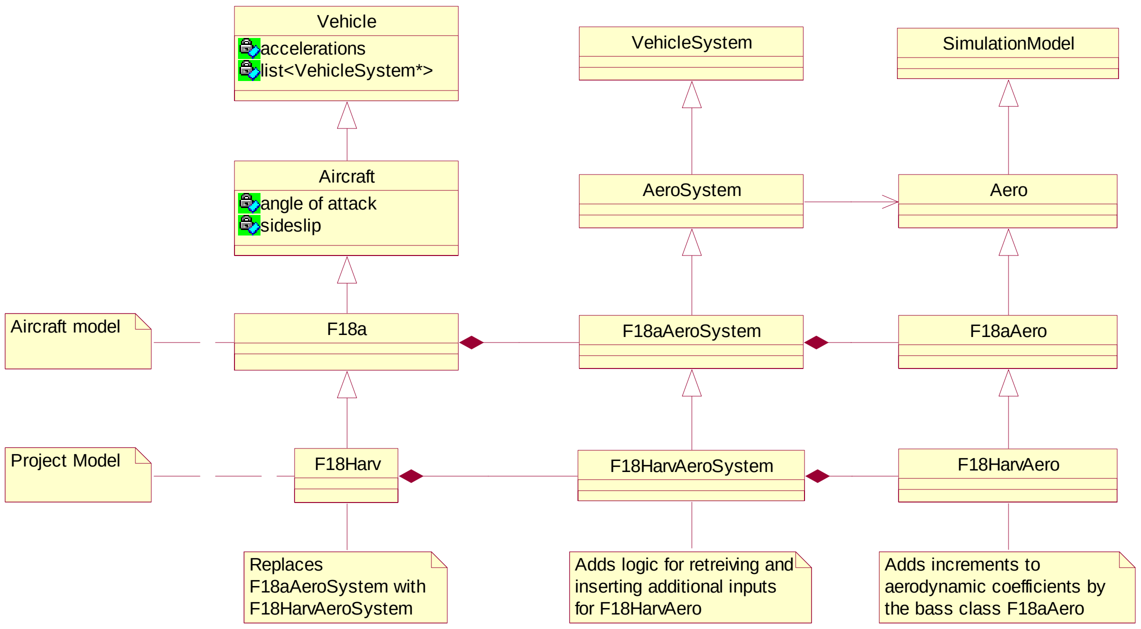

Figure 7. Vehicle Hierarchy

To create a credible Mars Scout proposal, the team needed to provide a detailed aircraft design and demonstrate that it could complete the mission objectives. As the design evolved, aircraft capability needed to be evaluated. The LaSRS++ application framework was used to support the aircraft design process. The framework allowed rapid prototyping of the aircraft as the design and mission parameters were modified. Specifically, the ARES simulation was developed to evaluate the flight of the aircraft starting from where the aircraft was fully deployed. Fully deployed is defined as free from the entry vehicle and unfolded with the drogue chute still attached. Other simulation tools were selected to model the entry body dynamics and aircraft unfolding. The LaSRS++ based simulation was therefore required to initialize the aircraft from the outputs of the other simulation tools, perform the pullout maneuver, and then navigate following the planned flight profile.

The ARES aircraft model was initially constructed with a minimal set of components. Aerodynamic, propulsion, and flight control system components were leveraged from other simulation projects and configured to ARES specifications. The mass properties, fuel system, sensor system, and other system components were constructed from mature, high fidelity framework models developed for other aircraft simulation projects. The leveraged simulation components were replaced with ARES specific models as the ARES models became available. Later, each ARES model would be replaced with progressively more detailed models. This allowed for performance evaluations as the design was maturing. Figure 8 illustrates the software components of the ARES vehicle.

The diagram shows that the ARES model was composed of an Ares object and all of its VehicleSystem objects. The system classes contained models through aggregation and/or inheritance. Classes used by the system through aggregation were typically unit-testable, meaning the model could be modified and fully tested before being integrated back into the simulation. This capability allowed the ARES simulation to rapidly absorb new changes and allowed the performance of the aircraft to be analyzed in a timely manner.

C. Integration of LaSRS++ ARES Simulation into CSIM

Integrating the LaSRS++ ARES simulation with CSIM presented an interesting set of challenges. As with any such integration, it is necessary to extract the core functionality of one simulation from its host environment and duplicate its output in another environment. This requirement can often be achieved simply by properly moding and scheduling the extracted code in the new simulation framework. In general, and in the case of this integration effort, the extracted code is modified as little as possible to ensure that the rehosted simulation functions properly. For this integration effort, simply executing the LaSRS++ simulation at the correct mode and rate was insufficient. In addition to duplicating the functionality of the LaSRS++ simulation, the integrated CSIM-LaSRS++ simulation required visibility and control over the data within the LaSRS++ code. Satisfying this requirement presented the steepest challenge of this effort because of the different simulation framework architectures.

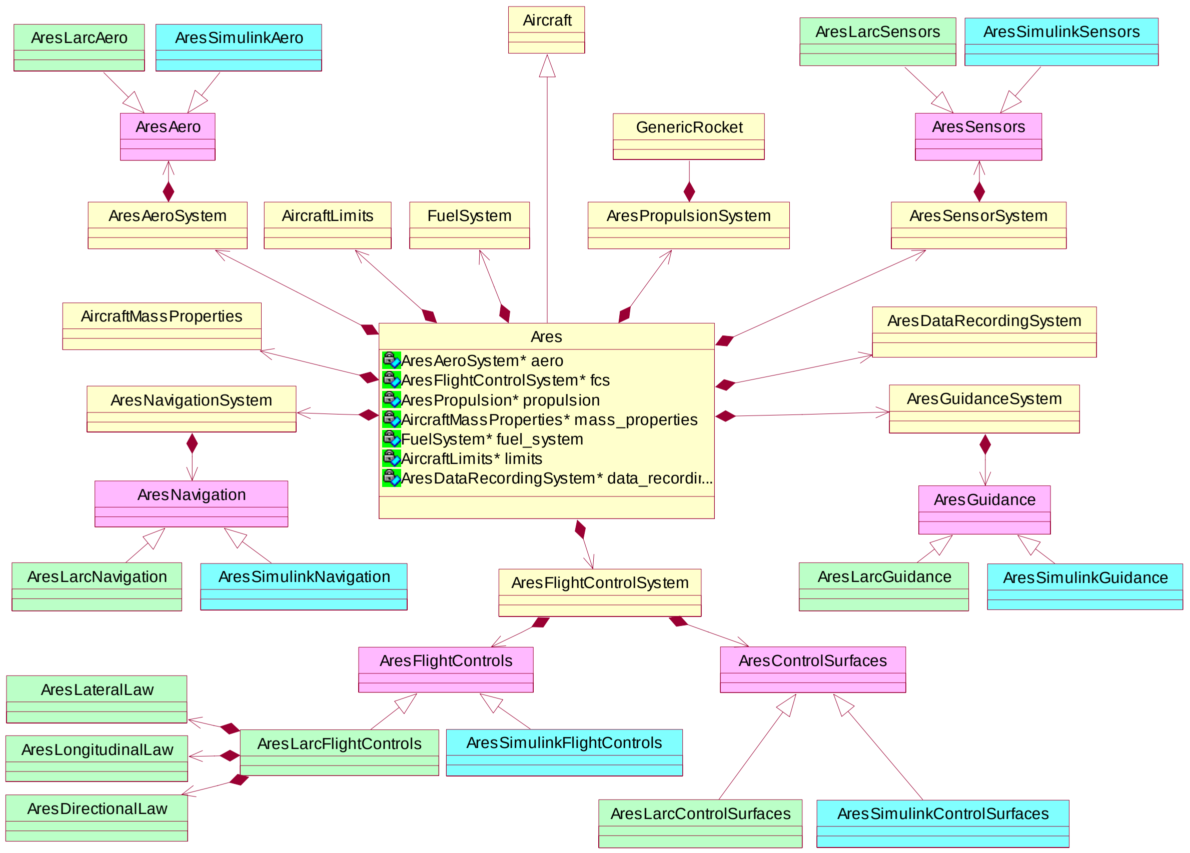

Figure 8. ARES Simulation Hierarchy

One of the core capabilities provided by any simulation framework is the ability to visualize, access, and manipulate simulation data at run time. Independently, both the LaSRS++ and CSIM frameworks provide these features. However, the LaSRS++ user interface was removed in order to integrate it into CSIM. Visibility and access to simulation data in CSIM is provided through a graphical browse window interface. Data in the CSIM simulation database are organized in statically allocated global structures that are arranged in a tree hierarchy. The data are viewed and manipulated by opening the desired directory in a browse window. Providing access to the LaSRS++ ARES simulation data from within the CSIM framework was problematic because the LaSRS++ framework is written in an object-oriented language (C++). A LaSRS++ simulation dynamically constructs itself at runtime based on the contents of an initialization file. Data that are available in the LaSRS++ simulation is not defined at compile time like that of a CSIM simulation, but are specified at initialization. Because of the flexible nature of the LaSRS++ framework, a flexible solution was required to incorporate its data into the static CSIM database.

The solution that was developed for this integration effort involved two layers. First, a CSIM application programming interface that enabled the user to dynamically add directories and variables to the database was written in C. Next, the CSIMDirectory base class utilizing this interface was written in C++. This class included pure virtual methods to map CSIM data into the LaSRS++ simulation and to map data from LaSRS++ objects into the CSIM database. Each newly allocated instance of this base class used the C interface to create a CSIM directory and inserted it into the CSIM database hierarchy at a specified location. From this base class, numerous child classes were derived, each specifically designed to map data between the CSIM directory that it created and a specific type of LaSRS++ object to which it maintained a pointer. Because these classes were designed to encapsulate a data directory, the classes themselves were hierarchical and included subdirectories.

At runtime, once the LaSRS++ ARES simulation had been constructed and initialized inside of CSIM, a pointer to the LaSRS++ Universe was passed to the constructor of one of these derived CSIMDirectory classes, the CSIMUniverse. That constructor created a CSIM database directory for the Universe data, and then created subdirectories for each World and Model in the Universe. The constructor for each derived CSIMDirectory class mapped the relevant LaSRS++ model data then created CSIM subdirectories for each of the major components of that model. This process continued recursively until each major object in the LaSRS++ simulation was mirrored by an instance of the appropriate derived CSIMDirectory class. As these CSIMDirectory derived objects were created, they were organized in a structure that mimicked the organization of the objects in the LaSRS++ simulation. Likewise, the CSIM database directories that these objects create were arranged with the same hierarchy. Once this process was completed, all data of interest in the LaSRS++ simulation were published in the CSIM database. This allowed the CSIM user to view and manipulate the LaSRS++ ARES simulation data at runtime.

After the LaSRS++ simulation was constructed, and its data had been published in the CSIM database, the integrated CSIM-LaSRS++ simulation was ready to execute. Scheduling and execution of the LaSRS++ ARES simulation inside of CSIM was handled through a top level function call that was inserted in the main CSIM simulation loop. This function handled moding of the LaSRS++ simulation, and ensured that it was executed at the proper rate. This function also handled all data mapping between LaSRS++ and CSIM. Upon entry into this function, the LaSRS++ simulation was set to the appropriate execution mode. This function then recursively traversed the tree of CSIMDirectory objects, mapping all inputs from the CSIM database into the LaSRS++ simulation objects. The function then called the top level LaSRS++ execute method. The function then recursively traversed the tree of CSIMDirectory objects to map all LaSRS++ simulation outputs to the CSIM database. In order to verify the integration of the LaSRS++ ARES simulation, the outputs of the CSIM-LaSRS++ simulation were compared against those of the standalone LaSRS++ simulation and no discrepancies were found.

The hardware interface capabilities of LaSRS++ were used to communicate data to and from the IO thread. Objects derived from the LaSRS++ HardwareInterface and HardwareDriver parent classes[9] were created and were registered with the LaSRS++ simulation to be called when the ARES simulation is executed. In the case of the HILSIM, the hardware driver code wrote and read to and from shared memory as opposed to a device driver. Using the existing LaSRS++ classes took advantage of the calling sequence within the LaSRS++ simulation to guarantee that data from the IAU was passed into the simulation at the correct point during model execution. The existing ARES code provided by LaRC was modified in order to execute in either a Software-In-The-Loop (SWIL) or HWIL mode. If configured for HWIL, the simulation software directed inputs from the IAU to the models for the actuator dynamics. Outputs from sensor models were also directed to the IAU via the HardwareInterface.

IV. Results from Simulation and Flight Software Integration

This section outlines the HILSIM integration efforts and lessons learned during the integration of the HWIL simulation and the IAU.

During simulation development, the simulation was executed in a SWIL mode prior to the availability of the IAU and FSW. The existing GN&C code provided with the LaSRS++ ARES simulation was used to control the airplane. Although the GN&C algorithms developed for execution on the IAU were different, the LaRC provided GN&C code allowed simulation developers to exercise the simulation much earlier in the program. This exercise helped verify that the LaSRS++ simulation had been correctly integrated within the CSIM simulation framework. Check cases were provided by LaRC that were used to verify the execution of the LaSRS++ simulation with CSIM.

The Draper GN&C code was developed in a separate desktop simulation of the ARES airplane before being integrated into the FSW to execute on the IAU. The development of this desktop simulation and the HILSIM were independent efforts with minimum collaboration. Integration issues were caused by the differences in the gravity, terrain, and world shape models since the GN&C code used the same models onboard as in the desktop simulation. These differences in the simulation environments resulted in unexpected behaviors during the simulated flight in the HILSIM. Once these issues were isolated and addressed, the airplane executed its mission as expected.

A lesson learned from this effort was that GN&C developers and testers should share a common set of environment models with defined parameters captured in a simulation requirements specification. The model differences between the GN&C development simulation and the HILSIM led to delays in integration. One positive outcome was that the differences made apparent the sensitivity of the GN&C algorithms to differences between the truth environment and the predicted environment of the onboard software. Once the two sets of models were in synch, differences in the truth and onboard models could be introduced to stress the algorithms in a controlled manner.

In addition to the mission IO, one of the serial interfaces between the simulation and the IAU was configured to output telemetry from the IAU to provide insight into the operation of the FSW. This capability was implemented only for debugging the integration of the IAU within the HILSIM and proved to be invaluable in determining the sources of error during integration. The IO serial data logger was configured to capture data from the FSW for later analysis. FSW developers could modify the contents of the telemetry data stream prior to each simulation run in order to provide the desired insight into the operation of the FSW. Because of bandwidth limitations of the serial data link, not all the FSW states could be telemetered at once.

Initial integration efforts also revealed that MIMU data were being lost due to the simulation and FPGA outputs not being synchronized. As explained above, this was solved by scheduling the execution of the simulation with the 200 Hz outputs of the FPGA MIMU simulator.

Another lesson learned was that tighter coordination between LaRC and Draper on simulation updates would be required on the Mars Scout program. The initial LaSRS++ ARES simulation was provided to Draper at the beginning of the PARR program and occasional updates were provided afterwards. Draper had no insight into changes being made to the ARES simulation. Although it was desired that LaRC and Draper would continue to share models and simulation updates throughout the PARR program, this was not successfully implemented. This issue was identified and was in the process of being addressed in order to support the Phase A proposal work. A revision control repository was being implemented at LaRC that Draper could have remote access to in order to receive the latest updates of the ARES simulation and to update the high fidelity sensor models being developed at Draper.

V. Conclusion

The ARES mission was not funded for Phase A concept study for the Mars Scout 2011 mission and the HILSIM effort was halted. Prior to stopping work on the HILSIM, the FSW executing on the IAU was successfully flying the ARES mission under ideal conditions. In addition, it was demonstrated that the IAU was capable of the throughput required for the science data for the proposed mission. Efforts were underway for a design of experiments to stress the operation of the GN&C algorithms. Error models for all sensors were developed in order to corrupt the measurements from the simulation as well as errors in the initial state knowledge of the GN&C algorithms. In addition, a tool was created in order to generate various wind profiles that would be expected during the plane's flight in order to further stress the operation of the GN&C code.

The HILSIM proved to be a successful integration of two simulation frameworks. The porting of the LaSRS++ simulation into CSIM framework saved resources by avoiding redundant efforts and taking advantage of the insight of the airplane development team at LaRC. The HILSIM would have provided a valuable resource in the future design of the ARES mission and had already begun to provide some insight into the requirements for flying an autonomous airplane on Mars.

A roadmap was also being developed to extend the simulation of the mission in HWIL prior to the pullout of the airplane and starting from separation from the spacecraft through entry and descent. This HWIL simulation would be required for further development of the FSW to include the other portions of the mission prior to the airplane flight. In addition, lessons learned from integrating the GN&C code with the HILSIM were also compiled for use on future programs similar to the ARES effort.

Acknowledgments

S. E. Slagowski and J. E. Vican would like to thank the following Draper Simulation Lab staff members for their work and input: Mike Curry, Kevin Quigley, and Stephen York. In addition, we would also like to thank the other Draper Laboratory PARR team members: William Wyman, Linda Fuhrman, Jeff Zinchuk, H. Lee Norris, Stan Shepperd, Stephanie Chiesi, Lisa Bodine, Joseph Kochocki, Patrick Battstone, Emily Pollock, and Bog Sniezek.

P. S. Kenney would like to thank the following Unisys Simulation Software Developers for their work and input: Susan Carzoo, Jason Neuhaus and Al Ragsdale. Special thanks go to Mark Croom and Henry Wright for all of their contributions to the ARES/PARR projects.

References

[1]Guynn, M. D., Croom, M. A., Smith S.C., Parks, R.W., Gelhausen, P.A., “Evolution of a Mars Airplane Concept For the ARES Mars Scout Mission”, *AIAA “Unmanned Unlimited” Systems, Technologies, and Operations – Aerospace, Land, and Sea Conference and Workshop & Exhibit*, September 2003, San Diego, CA, AIAA-2003-6578.

[2]Levine, J.S., et al., “Science From a Mars Airplane: The Aerial Regional-Scale Environmental Survey (ARES) of Mars”, *AIAA “Unmanned Unlimited” Systems, Technologies, and Operations – Aerospace, Land, and Sea Conference and Workshop & Exhibit*, September 2003, San Diego, CA, AIAA-2003-6576.

[3]Sandford, S.P., et al., “ARES and Beyond: Autonomous Aerial Platforms Create a Unique Measurement Capability for Earth and Planetary Science”, *AIAA “Unmanned Unlimited” Systems, Technologies, and Operations – Aerospace, Land, and Sea Conference and Workshop & Exhibit,* September 2003, San Diego, CA, AIAA-2003-6610.

[4]Wright, H.S., Croom, M.A., Braun, R.D., Qualls, G.D., Levine, J.S., “ARES Mission Overview – Capabilities and Requirements of the Robotic Arial Platform”, *AIAA “Unmanned Unlimited” Systems, Technologies, and Operations – Aerospace, Land, and Sea Conference and Workshop & Exhibit,* September 2003, San Diego, CA, AIAA-2003-6577.

[5]Kenney, P. S., “Rapid Prototyping of an Aircraft Model in an Object Orientated Simulation”, *AIAA Modeling and Simulation Technologies Conference and Exhibit*, August 2003, Austin, TX, AIAA-2003-5816.

[6]Kenney, P. S., “Simulating the ARES Aircraft in the Mars Environment”, *AIAA “Unmanned Unlimited” Systems, Technologies, and Operations – Aerospace, Land, and Sea Conference and Workshop & Exhibit*, September 2003, San Diego, CA, AIAA-2003-6579.

[7]Kenney, P.S., Geyer, D.W., “Using Abstraction to Create a Portable Object-Orientated Simulation”, *AIAA Modeling and Simulation Technologies Conference and Exhibit*, August 1999, Portland, OR, AIAA-1999-4340.

[8]Maden, M. M., “Architecting a Simulation Framework for Model Rehosting,” *AIAA Modeling and Simulation Technologies Conference and Exhibit*, August 2004, Providence, RI, AIAA-2004-4924.

[9]Kenney, P. S., Leslie, R. A., Geyer, D.W., Madden, M.M., Glaab, P.C., and Cunningham, K., “Using Abstraction to Isolate Hardware in an Object-Oriented Simulation”, *AIAA Modeling and Simulation Technologies Conference and Exhibit*, August, 1998, Boston, MA, AIAA-98-4533.