NASA Engineering & Technology Directorate

Control & Data Systems Division

Application & Simulation Software Engineering Branch

[NE-C1]

Supporting Multiple Programs and Projects at NASA's Kennedy Space Center

Camiren L. Stewart

GS-07, Level 01

Electrical Engineering, Masters of Engineering

University of Cincinnati, Cincinnati, OH.

Fall, 2014

Supporting Multiple Programs and Projects at NASA's Kennedy Space Center

Camiren L. Stewart[1]
University of Cincinnati, Cincinnati, Ohio, 45220

Nomenclature

AGSM	=	Advance Ground Systems Maintenance
CAIDA	=	Customer Avionics Development & Analysis
CCP	=	Commercial Crew Program
CM	=	Content Management
CMS	=	Content Management System
COTS	=	Commercial off the Shelf
CUI	=	Compact Unique Identifier
CxP	=	Constellation Program
DPI	=	Default Physical Interface
GIS	=	Ground Integrated Schematic
GSDO	=	Ground Systems Development and Operations Program
GOTS	=	Government off the Shelf
GSE	=	Ground Support Equipment
GUI	=	Graphical User Interface
I/O	=	Input / Output
ISS	=	International Space Station
JSC	=	Johnson Space Center
KSC	=	Kennedy Space Center
LCC	=	Launch Control Center
LCS	=	Launch Control System
LTS	=	Long Term Support
LEO	=	Low Earth Orbit
MAESTRO	=	Managed Automation System for Test, Real-time Operations
MPCV	=	Multipurpose Crew Vehicle
MSFC	=	Marshall Space Flight Center
NASA	=	National Aeronautics & Space Administration
OS	=	Operating System
PLC	=	Programmable Logic Controller
PSP	=	Pilot Support Patch
RTW	=	Real-Time Workshop
SAGE	=	Scalable Adaptive Graphics Environment
SIM	=	Simulation
SLS	=	Space Launch System
SCCS	=	Spaceport Command and Control System
SMS	=	System Mechanical Schematic
SWORDS	=	Soldier-Warfighter Operationally Responsible Deployer for Space
UPSS	=	Universal Propellant Servicing System

[1] KSC Pathway Intern, NE-C1, Kennedy Space Center, University of Cincinnati

I. Introduction

With the conclusion of the shuttle program in 2011, the National Aeronautics and Space Administration (NASA) had found itself at a crossroads for finding transportation of United States astronauts and experiments to space. The agency would eventually hand off the taxiing of American astronauts to the International Space Station (ISS) that orbits in Low Earth Orbit (LEO) about 210 miles above the earth under the requirements of the Commercial Crew Program (CCP). By privatizing the round trip journey from Earth to the ISS, the space agency has been given the additional time to focus funding and resources to projects that operate beyond LEO; however, adding even more stress to the agency, the premature cancellation of the program that would succeed the Shuttle Program – The Constellation Program (CxP) –it would inevitably delay the goal to travel beyond LEO for a number of years.

Enter the Space Launch System (SLS) and the Orion Multipurpose Crew Vehicle (MPCV). Currently, the SLS is under development at NASA's Marshall Spaceflight Center in Huntsville, Alabama, while the Orion Capsule, built by government contractor Lockheed Martin Corporation, has been assembled and is currently under testing at the Kennedy Space Center (KSC) in Florida. In its current vision, SLS will take Orion and its crew to an asteroid that had been captured in an earlier mission in lunar orbit. Additionally, this vehicle and its configuration is NASA's transportation to Mars.

Engineers at the Kennedy Space Center are currently working to test the ground systems that will facilitate the launch of Orion and the SLS within its Ground Services Development and Operations (GSDO) Program. Firing Room 1 in the Launch Control Center (LCC) has been refurbished and outfitted to support the SLS Program. In addition, the Spaceport Command and Control System (SCCS) is the underlying control system for monitoring and launching manned launch vehicles.

As NASA finds itself at a junction, so does all of its associated centers across the US. KSC has found itself at the blunt end of change as the entire center has transitioned from an operations mindset to a development mentality. The author of this paper has had the fortunate privilege and opportunity to be part of a transforming NASA during the fall months of 2014. The following is a high level account of projects that he had the chance to work on including the Spaceport Command and Control System, the Advanced Ground System and Maintenance Program Project, Customer Avionics Development & Analysis (CAIDA) Lab and Swamp Works.

II. SCCS Models and Simulation

Real-time simulations of ground support equipment (GSE) and launch vehicle systems are required throughout the life cycle of SCCS to test software, hardware, and to engineer procedures for launch team training. The simulations of the GSE at the launch site in conjunction with off-line processing locations are developed using Simulink, a piece of Commercial Off-The-Shelf (COTS) software. The simulations that are built are then converted into code and run in a simulation engine called Trick, a Government off-the-shelf (GOTS) piece of software developed by NASA's Johnson Space Center (JSC). The Simulation and Product group contains a two-part team in which one side builds the simulations needed to stand in place of the GSE and the other side that lays the framework needed for the models to be controlled and respond seamlessly when viewed from the Launch Control System (LCS).

Models and Simulation

Specifically, these simulations are built to closely match the engineering Ground Integrated Schematics (GIS), which contains the electrical portion of the targeted GSE, and the System Mechanical Schematic (SMS), which contains the cryogenic and pneumatics portions of the targeted GSE. The functionality, logic, and even picture behind each component is built into custom Simulink library components and then tied together to form the GSE simulation. It is therefore evident that these custom Simulink library components make up the majority of the products that the SCCS Simulation group produces.

In the world of software, tools that are used to create products are inevitably upgraded by their specified vendor to increase performance, add functionality, and provide overall improvements that the vendor sees fit; Mathwork's Simulink simulation[2] software is no exception. Unfortunately, the process of upgrading the vendor software is not sufficient to complete the conversion. In the SCCS simulation team's world, the upgrade would be from Matlab R2009b to Matlab R2013a, a jump of seven versions. Many features have been added, deprecated, or altered to the point where a product that was built in R2009b may not function the same way in R2013a. Therefore, the task was given to upgrade each individual library component and ensure there is no delta between the uses of each component between both Matlab versions.

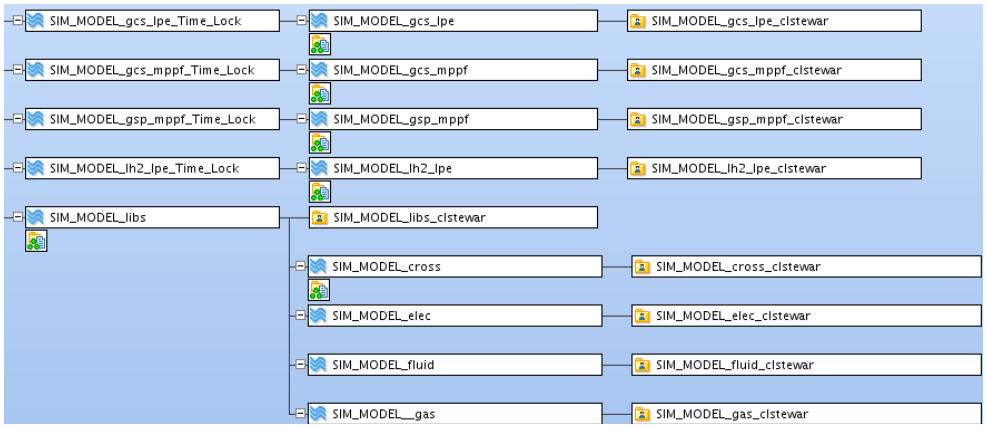

Figure 1 - AccuRev Stream Architecture

The majority of the path-finding and code writing was completed between February 2014 and July 2014 and its details are referenced from *Upgrading Custom Simulink Library Components for use in Newer Versions of Matlab*, a report written by Camiren Stewart and was published July 18th, 2014 (Stewart, 2014). This particular portion will cover the days leading up to the upgrade execution, the upgrade itself, and mitigations following the upgrade. The following will pick up where that report left off.

One major issue identified prior to the upgrade was that of how the AccuRev streams were architected. The library component stream, SIM_MODEL_libs, where almost all of the library components are developed and tested, and is purposely separated from each subsystem stream. *Error! Reference source not found.* shows an excerpt from a visual representation of how the streams are laid out in the development repository. Theoretically, as components were developed and tested in the SIM_MODEL_libs streams, those components would be released to "flow downstream" and appear in the GSE subsystem streams below SIM_MODEL_libs. This would create new instances of that component in each subsystem stream. A potential issue could arise in the event where if developer were to create a subsystem workspace and modified a library component inside that workspace, then the library component that was modified would not receive the upgrade patches and would instead keep the modifications that the developer had previously performed. This is a safety net built into the Content Management System (CMS) – AccuRev – that is in use. To mitigate this risk, the subsystem streams were XLINKED (the same idea of a Windows shortcut or Linux Symbolic link) inside of the AccuRev environment. This would create an environment where the library component can only be modified in SIM_MODEL_libs and only a read instance of these library components exist in each subsystem stream. This consolidation not only simplifies the AccuRev structure in a manner where it is more manageable, but it also means that the upgrade would only need to be targeted to the SIM_MODEL_libs stream ensuring that all of the custom library components in the subsystem streams catch the upgraded components.

The upgrade itself was planned to be implemented over a weekend to minimize the impact it would have on library component development and testing. During the upgrade process, SIM_MODEL_libs was locked down to prevent any development, but the subsystem streams remained available. Subsystem layout was not affected by the Matlab upgrade since the library file path and component name in the library reference were not to be changed; however, the idea was to successfully complete the upgrade in the quickest manner possible while maintaining completeness with accuracy. The libraries inside of SIM_MODEL_libs (electrical, fluid, and gas) were upgraded individually. Fortunately, most of the faults that were presented were worked out prior to the upgrade and therefore enabled the

[2] http://www.mathworks.com/products/simulink/

actual upgrade of the custom library components to finish rather quickly. During the upgrade, a number of the components had logic that was changed internally to account for new errors introduced in the new version of Simulink that would inadvertently halt the simulation. This is an unwanted result in the production environment and therefore needs to be accounted for. Due to these internal logic modifications, the components that were either released or the components that were currently in a review were required to be regression tested. Simply put, the upgraded component would be tested using the same unit test that was used to originally test the requirements of the component. If the upgraded component showed zero deltas between the R2009b component and the R2013a component, the component would be considered fully upgraded and ready to be published to the production environment. If deltas occurred that could not be explained, then additional modifications would have to be implemented until the violating component passed the regression test. ***Error! Reference source not found.*** details the estimated time it took for the upgrade to complete including the regression tests. Where the upgrade itself was relatively painless, most of the time spent during the execution was running the regression tests and analyzing the results. The time that was spent fixing regression tests that did not pass was also included into the calculated time.

DOMAIN	UPGRADE	REGRESSION TESTING
ELECTRICAL	39 Minutes	67.5 Hours
FLUID	21 Minutes	18.0 Hours
GAS	10 Minutes	19 Minutes

Table 1 - Upgrade and Regression Testing Time Estimates

The component upgrade and regression testing was where the bulk of the upgrade work resided; however, there were many other bits and pieces that were required to be worked out in order to label the upgrade a success. One of these tasks was with a portion of code that Mathworks supplies to the SCCS Simulation group. The Trick Pilot Support Patch (PSP) is a piece of code that is installed in-line with Matlab and provides the option for Simulink to have Trick as a target device. Because this was a product that was supplied by Mathworks, it took a number of hours to work with the company and their engineers to come up with a newer, improved PSP that would support Matlab R2013a and the version of Trick that NASA was currently using.

On the same lines of working and coordinating with groups outside of the Simulation and Modeling Group, it was discovered post-upgrade that the build server – the device used to formally create the catalog of Models, SIM Framework and addition Launch Control System (LCS) products – would need to be configured appropriately in order to accept the new Matlab R2013a simulation models. The coordination effort included a number of people to fully obtain a working build server with the new version of Matlab. The goal would be to create the same environment on the build server that was on the local development machines in terms of Matlab. That means that the new version of Matlab (R2013a) and the Trick PSP would need to be reviewed and checked into Content Management (CM) before the products could be deployed into the production environment.

Framework

The Simulation Framework consists of the items needed to trick LCS into thinking that the real-world end items are present. SoftLogix Programmable Logic Controller (PLC) Emulator takes place of the PLC's and the PLC logic that would be deployed on the appropriate PLC is loaded into SoftLogix. Additionally, as of September 2014, NASA's Marshal Space Flight Center (MSFC) in Huntsville, Alabama has delivered the SLS simulation and its purpose will be to use it in conjunction with the Orion simulation and the GSE simulations. *Figure 2 - SCCS Simulation Overview* gives a high level representation of the project.

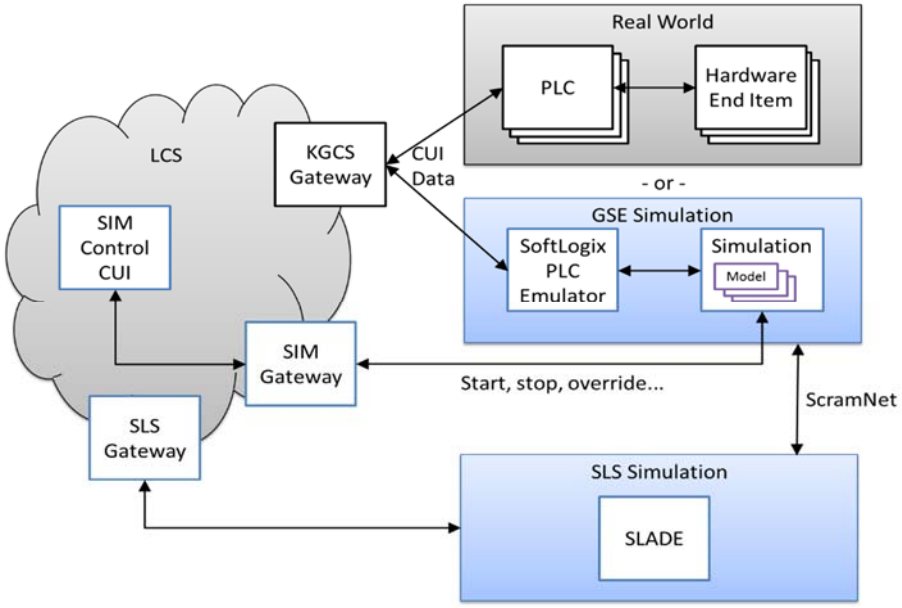

Figure 2 - SCCS Simulation Overview

Now that the SLS simulation has been delivered, it needs to be integrated into the SCCS architecture. This includes many amendments to the framework that is already deployed to the production environment. One of these items is the ability to choose the path to the configuration file and its supporting configuration files for the SLS simulation product. Due to CM restraints, the modification of the original configuration file is prohibited; therefore, it was determined that the framework group would need to add in the ability to load their own tailored configuration code into the source code.

Inside of the LcsEmulator TrickWrapper, there is source code, written in C++, which enables the loading of this configuration file named SIM_Utility_Config.xml. To give the option for the operator to use their own configuration file, the source code was modified to check if two Linux system variables exist in the environment: *GLADE_CONFIG_PATH* and *GLADE_CONFIG_FILE*. If these two items exist, then it will load the operator's configuration file instead of the default. Error checking was also implemented so that if an invalid value is set for either of these two variables, then it would default to the primary configuration file. If the two variables have acceptable values, then the *GLADE_CONFIG_FILE* in the *GLADE_CONFIG_PATH* is loaded. Additionally, any additional file that is found in the directory of the value in *GLADE_CONFIG_PATH* is also loaded into the configuration, overwriting the default variables in the original configuration. All of this work is strictly contained to one file in the source code: *LcsEmulatorWrapper.C.*

The second task for the SCCS Simulation Framework group was to replace the Simulation (SIM) Control Graphical User Interface (GUI) with Managed Automation System for Test, Real-time Operations (MAESTRO) Test Conductor. *Figure 2 - SCCS Simulation Overview* shows a SIM Control GUI located inside of the LCS architecture. This is a program, written in JAVA, which is used in the LCS architecture to manually override inputs and outputs (I/O) in the GSE models. This SIM Control GUI will be removed in future releases and instead will be replaced by the MASTEO Test Conductor. The Test Conductor will not live in the LCS environment, and instead will reside in the GSE Simulation environment on a Linux machine. This is forward work and has not been completed as of the time of this writing.

III. Advanced Ground System Maintenance (AGSM)

Under the Ground Systems and Development Operations (GSDO) Program, the Advanced Ground System Maintenance (AGSM) Project is focused on the broad goal of lowering the cost of operations and maintenance through

targeted investment in different opportunities throughout the launch preparedness and proceedings. This objective is accomplished through the application of integrated system health management components that range from Anomaly Detection (AD), Fault Isolation, Diagnostics using high-fidelity physics models, Prognostics, and more.

Simulations are especially important during the design and operation of engineering systems. These models serve in many forms; for example, the model can be used by model-based health management tools to develop prognostic and diagnostic models. With the integration of nominal and faulty behavior along with the ability to inject faults into the components of each simulation, these simulations can be used for operator training in nominal and off-nominal situations, and even developing and prototyping health management algorithms (Barber, Johnston, & Daigle, 2013). Additionally, this provides the engineering team the ability to validate and verify their design, giving them the opportunity to see the reaction of an injected fault at any point downstream from the anomaly, no matter where an official gauge or transducer is set. For example, in a given system, there may be a small number of transducers and gauges that are operating inside of the system that measure values such as flow rate, pressure, and more; however, a situation may occur where the need to know a value at a certain point in the system where there is no measurement device could happen. Using a high-fidelity model, the team can set the conditions in the model similar to the real-life scenario and can virtually place a sensor anywhere in the model.

Due to a reasons ranging from lack of resources to the recent inception of the simulation group in AGSM, the standards, libraries, and models are not as developed as the SCCS models. Therefore, it was decided to use the

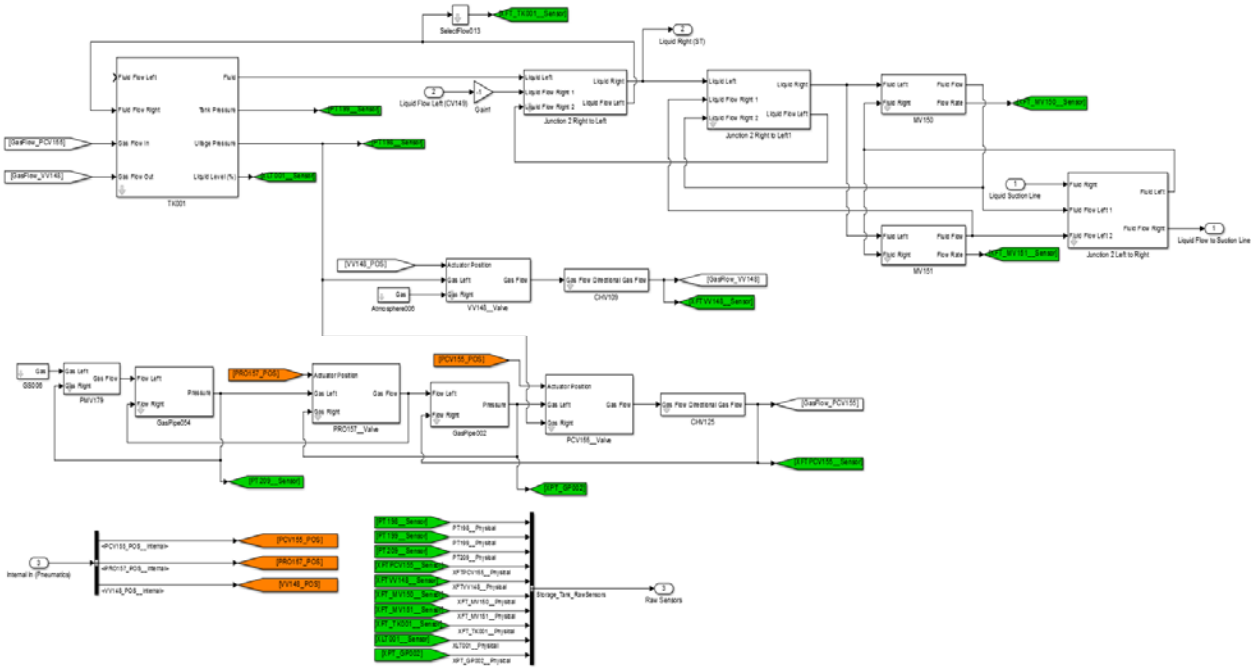

Figure 3 - Cryogenics Test Best Model

assistance of adopting SCCS standards into the AGSM library components; the result of combining the SCCS and AGSM techniques catapulted the AGSM component library into a better, more organized structure that will assist modeling subsystems substantially. For example, ***Error! Reference source not found.*** shows an excerpt from the model for the Cryogenics Test Bed (CTB) that is located in the Cryogenics Lab at KSC. The model shows a number of junctions, valves, and one tank. Here, each line that connects to each component is called a signal and represents I/O that enters the component, manipulated in some fashion, then exits the component. Single values travel in only one direction and is the reason why there are multiple inputs and outputs in each component. It is noticeable that the model is disorganized and hard to read, despite the best efforts of the engineer who integrated these components.

To better assist modeling efforts, the entire AGSM library was reconstructed to adopt some SCCS standards and common practices while obtaining some previous standards already implements. There were a number of reasons to pursue such a task, but the most common ones are because SCCS must adhere to more strict modeling standards set

at the project, center, and agency levels, the time spent path finding has already been applied to the multiple ways that a task can be completed. Additionally, it was originally thought that as both projects mature, AGSM and SCCS could perform information sharing between each group with minimal translation needed between each library component. Standards that were adopted were mostly administrative such as naming conventions and component mask utilization; however, one large functionality standard that was adopted made a large impact on the way AGSM modeled their simulations. This was the utilization of the Default Physical Interface (DPI).

The DPI is the basic building block of a library component. A custom Simulink library component, it provides values a two lane road for signals. Inside the DPI block, the Simulink signals representing the state variables of the component are bundled into a signal physical signal that are connected to each related neighboring component. With the ability to create a signal with values traveling in opposite directions, it simplifies the component and creates a more organized model. Where there were multiple input and output points on a custom library block, it now typically only has 2 points for signal lines to be connected.

Shown in *Figure 4 - UPSS Schematic* Figure *5 - UPSS* is the new translation between a SMS and the Simulink model with DPI's implemented in the library components. Instead of having signals arbitrarily routed throughout the model, like *Figure 3 - Cryogenics Test Best Model*, signals are now seamlessly integrated between components and better represents the original schematic.

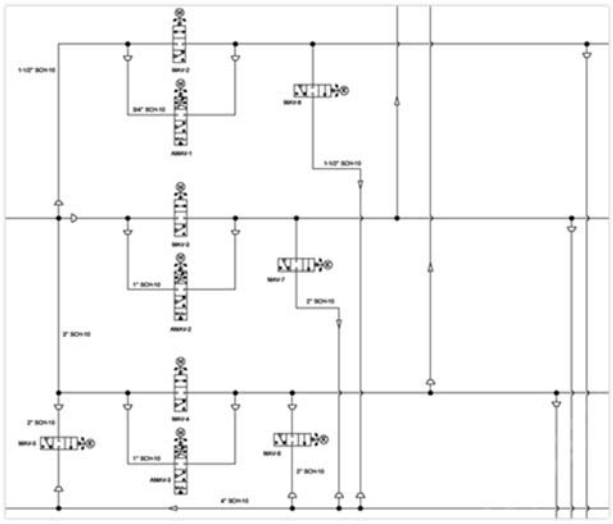

Figure 4 - UPSS Schematic

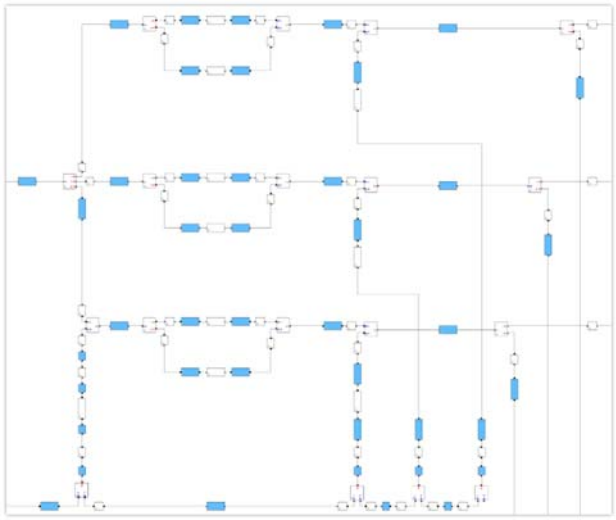

Figure 5 - UPSS Simulation Model

Now that the library component had been upgraded and were easier to work with, they could now be integrated into a simulation model that represent a subsystem. The target project, Soldier-Warfighter Operationally Responsible Deployer for Space (SWORDS) launcher is a joint project between NASA, U.S. Army Space and Mille Defense Command/Army Forces Strategic Command, and the Office of the Secretary of Defense. The purpose of which would be to deploy nanosatellites into space with exact precision and low cost by enabling combatant commands to have a launch-on-demand capability (USASMDC/ARSTRAT, 2014).

Now part of the Advanced Explorations System (AES) program, the Universal Propellant Servicing System (UPSS) will provide a fuel and oxidizer propellant delivery system that can be used for small and medium class launch vehicles. With the aim of low cost, fast fueling objective, the system will be designed to be portable to provide the capability to service vehicles at multiple locations. AGSM, and specifically the author of this paper, would have the responsibility of modeling the UPSS. Currently still under development, an excerpt from the UPSS schematic and model are shown in *Figure 4 - UPSS Schematic* Figure *5 - UPSS Simulation*

Model. With the new upgraded library components, integrating them to model the UPSS is rather intuitive. After integrating them, mask parameters must be changed to better tailor each simulated component to its actual function.

Though each library is rather robust, inevitably there are components that exist in the subsystem schematic that does not currently reside in the AGSM library. This leads to the creation of a new component and is where most of the technical work is performed in the simulation. In many occasions, the component is interpreted and equations are based off of those interpretations by the developer. For complex components that might be unique or proprietary to the subsystem in the way in which they are configured, a Subject Matter Expert (SME) for the subsystme might be consulted that is knowledgeable about the system. There were a number of generic components that needed to be created in order to support the UPSS model; components from different domains, but have relatively the same functionality needed to build such as a gas flex hose, a fluid flex house, and components like gas junctions. The difference between the components in each domain is the DPI that is used and how the logic inside the component handles the commodity that is being passed through. What makes this practice a little challenging is the process and mixture between cryogenic and fluid domain components. It is difficult to model the transformation of gas to fluid and vice-versa, in addition to two-phase flow. As of the writing of this document, that work is currently underway, but when finished, will provide to give extremely high-fidelity to the UPSS model.

IV. Customer Avionics Development & Analysis (CAIDA) Lab

The Customer Avionics Development and Analysis (CAIDA) lab is used as an emulator of Orion's flight software and hardware. This is in turn used to support Orion and GSDO development and testing (Herridge, 2014). The CAIDA lab processes Orion's raw telemetry data, amount other functions. This data and its values are compacted into a unique string called a Compact Unique Identifier (CUI). To monitor the simulation data, a piece of COTS software is used called Dewesoft[3] to monitor the data from each CUI. Dewesoft enables the CAIDA lab and its operators the ability to examine the network data, outputs the results in a numerous unique, propriety ways, and save the displays for later references. A tool is used called the XML Display Generator that creates a display with multiple CUI references. This tool uses many inputs from the user and references a telemetry database to generate an XML file that is used by Dewesoft to display the CUI data.

During the time spent in AGSM, engineers were in need of data that was dependent on this XML Display Generation tool. Unfortunately, the tool was broken and needed to be fixed. This provided an opportunity for the author of this paper to extend his reach outside of AGSM and SCCS and into CAIDA. This also provided to be an excellent learning opportunity to get experience in writing C# as it was the language used to create the XML generator tool. Fortunately, it was a quick fix that only related to changing a few integers around, but the knowledge that was obtained with the overview of the CAIDA lab, Microsoft's Visual Studio Express 2013, and the C# language proved to be an excellent learning experience.

Additionally, during the time devoted to work in the CAIDA lab, it was noticed that there was no real way that artifacts, such as the XLM Display Generator, that were produced were under any form of version control. Instead of one instance of the source code for the program in existence, there were multiple instances on different locations throughout the CAIDA lab. This provided an opportunity to establish a local CMS with the assistance of Git.

Git is a multiplatform distributed version revision control system that is free to use and distributed under the terms of the Generic Public License. For these reasons, it was an excellent candidate to solve the lack of version control software that the CAIDA lab had acquired. Using a multi-terabyte Network-Attached Storage (NAS) device, a local Git server was installed on a local Linux box and housed its repositories on the NAS. This configuration provided to be a great solution to house and track the software that is developed in the CAIDA lab.

[3] http://www.dewesoft.com/

V. Swamp Works

The Swamp Works Laboratory at KSC is home to some of the most technically innovative projects that KSC has to offer. Named after the Lockheed Martin legendary Skunk Work's that produced projects such as the U-2 spy plane and F-117 stealth fighter, KSC named it Swamp Works due to its similar characteristics to Skunk Works and the location in Florida – that it's located in the middle of a swamp (Siceloff, 2013). The ideas and projects that are passed around the Swamp Works Laboratory are unique in a way that they are not hindered by the typical government course of work that involves large amount of process and paper work. The projects developed in the lab are involved in rapid prototyping, an extreme amount of creativity, and are meant to push the bounds of innovation with minimal resistance of outside sources.

The Hyper Wall inside of the Swamp Works Lab is no exception. In what started out as a kick start proposal in 2013, the Hyper Wall is a prototype device that will be used as a pathfinder project for telepresence and collaboration at conferences and meetings in and around the Kennedy Space Center. Due to its mobile capacity and ability to display high resolution video, the Hyper Wall will provide to be an invaluable tool when processing visual data and will enhance the NASA experience in a presentation & outreach manner wherever it is used.

The Hyper Wall is composed of both new and old hardware. The computers and monitors used to drive the Hyper Wall is from repurposed firing room devices from the Shuttle Program. The only additional hardware that was purchased were the video cards, the *AMD FirePro W600*[4]. This card would provide the ability to handle the output of six monitors during the proper configuration.

With the hardware setup complete, Ubuntu 14.04 Long Term Support (LTS) was loaded onto each machine. The free distribution and high performance of Linux made this a great operating system (OS) to be the driver of the Hyper Wall. The guts of the Hyper Wall is controlled using a piece of software called Scalable Adaptive Graphics Environment (SAGE). Developed at the University of Illinois, SAGE is a cross-platform, open-source middleware that enables users to have a common operating system to access and display content to one or more tiled display walls. This makes it a perfect candidate to integrate the Hyper Wall displays in Swamp Works.

The skills obtained during the work on the Hyper Wall project in Swamp Works were invaluable. Due to its dependency on Linux, the author of this paper had the opportunity to expand on his Linux knowledge and problem solving skills that were required. Additionally, the hardware and software used were both new concepts to grasp that will have the opportunity to prove themselves beneficial for projects in the future.

VI. Conclusion

One of NASA's strategic plan is to expand the frontiers of knowledge, capability, and opportunity in space. There is no doubt that the Kennedy Space Center is supporting this within its Ground Support and Operations Program, and specifically within AGSM, the CAIDA Lab, and SCCS. These projects are all developed to directly support the vision of a multi-user spaceport at KSC. After the successful development of the vehicle and launch control system, the United States will continue to extend its reach into space and grasp the limitless knowledge and exploration it has to offer.

In addition, the Hyper Wall project within Swamp Works is being developed as part of a Research and Technology Innovation effort at KSC, which directly supports the development of technologies to improve the quality of life on our home planet within the NASA strategic plan.

[4] http://www.amd.com/en-us/products/graphics/workstation/firepro-display-wall

VII. Acknowledgments

The author of this paper would like to sincerely thank Cheryle Mako and Lien Moore for providing their mentoring proficiencies to enhance my skills as an engineer. Their enthusiastic encouragement and persistent guidance were and continue to be a valuable quality that I have the gratification to be exposed to. The pair have proven many times over to be an invaluable asset to NASA as they continue to coach the mentees around them and strive for them to always perform at the highest tier.

Secondly, I would like to thank Jason Kapusta for his unlimited knowledge and boundless patience that he shared with me during the conceptualization of the framework and project within SCCS. His advice and collaboration is tough to put a value on, and it is my esteemed pleasure to work alongside someone with the level of intelligence that he has to offer.

Additionally, I would like to thank John Barber for sharing his vast comprehension with AGSM, and specifically the Anomaly Detection and Simulation projects. John has reaffirmed my belief that there is always something to learn, even in areas of education that one feels that they have mastered. He has a spectacular ability to give simple explanations to the most difficult of problems. Additionally, the ability that he has to create useful and constructive ideas in a team environment are incredible. Some of the most amazing products and solutions to problems that I have witnessed at NASA came from my time with John as my mentor.

Finally, I wish to thank Curtis Williams for his guidance and expertise within the CAIDA Lab. Curtis is one of the hardest working engineers that I have had the pleasure to work alongside. He has no issue with encouraging the young engineers around him which directly resulted in giving me the ability to come up with lofty solutions to complex problems.

VIII. Works Cited

Barber, J. P., Johnston, K. B., & Daigle, M. (2013). *A Cryogenic Fluid System Simulation in Support of Integrated Systems Health Management*. Retrieved from Phmsociety.org: http://www.phmsociety.org/sites/phmsociety.org/files/phm_submission/2013/phmc_13_023.pdf

Herridge, L. (2014, May). *NASA*. Retrieved from SpacePort Magazine: http://www.nasa.gov/sites/default/files/files/may2014v4.pdf

Siceloff, S. (2013, June 1). *Swamp Works Thrives on New Engineering Approach*. Retrieved from NASA: http://www.nasa.gov/centers/kennedy/news/masters-swampworks.html

Stewart, C. L. (2014, July 18). Upgrading Custom Simulink Library Components for use in Newer Versions of Matlab. Cape Canaveral, Florida, United States of America.

The MathWorks, Inc. (2014, July 01). *Platform Road Map for the MATLAB and Simulink Product Families*. Retrieved from MathWorks: http://www.mathworks.com/support/sysreq/roadmap.html

USASMDC/ARSTRAT. (2014, October 28). *SWORDS*. Retrieved from http://www.smdc.army.mil/FactSheets/SWORDS.pdf