Development and Testing of Functionally Operative and Visually Appealing Remote Firing Room Displays and Applications

Kristy Quaranto
NASA Kennedy Space Center
Major: B.S. Aerospace Engineering
KSC FO: Summer Session
July 22nd, 2014

Table of Contents

Development and Testing of Functionally Operative and Visually Appealing Remote Firing Room Displays and Applications

Kristy Quaranto[1]
Embry-Riddle Aeronautical University, Daytona Beach, FL 32114
Kennedy Space Center, FL 32899

I. Abstract

This internship provided an opportunity for an intern to work with NASA's Ground Support Equipment (GSE) for the Spaceport Command and Control System (SCCS) at Kennedy Space Center as a remote display developer, under NASA technical mentor Kurt Leucht. The main focus was on creating remote displays and applications for the hypergolic and high pressure helium subsystem team to help control the filling of the respective tanks.

As a remote display and application developer for the GSE hypergolic and high pressure helium subsystem team the intern was responsible for creating and testing graphical remote displays and applications to be used in the Launch Control Center (LCC) on the Firing Room's computers. To become more familiar with the subsystem, the individual attended multiple project meetings and acquired their specific requirements regarding what needed to be included in the software. After receiving the requirements for the displays, the next step was to create displays that had both visual appeal and logical order using the Display Editor, on the Virtual Machine (VM). In doing so, all Compact Unique Identifiers (CUI), which are associated with specific components within the subsystem, were need to be included in each respective display for the system to run properly. Then, once the display was created it was to be tested to ensure that the display runs as intended by using the Test Driver, also found on the VM. This Test Driver is a specific application that checks to make sure all the CUIs in the display are running properly and returning the correct form of information. After creating and locally testing the display it needed to go through further testing and evaluation before deemed suitable for actual use. For the remote applications the intern was responsible for creating a project that focused on channelizing each component included in each display. The core of the application code was created by setting up spreadsheets and having an auto test generator, generate the complete code structure. This application code was then loaded and ran on a testing environment set to ensure the code runs as anticipated.

[1] Software Developer for Remote Firing Room Displays and Remote Control Applications, Spaceport Command and Control System, Kennedy Space Center, Control and Data Systems Division NE-C1, Embry-Riddle Aeronautical University, Daytona Beach

By the end of the semester long experience at NASA's Kennedy Space Center, the individual should have gained great knowledge and experience in various areas of both display and application development and testing. They were able to demonstrate this new knowledge obtained by creating multiple successful remote displays that will one day be used by the hypergolic and high pressure helium subsystem team in the LCC's firing rooms to service the new Orion spacecraft. The completed display channelization application will be used to receive verification from NASA quality engineers.

II. Introduction

With the close of the 30 year long Space Shuttle program, NASA has now committed their efforts to explore beyond Earth's orbit and eventually to Mars. The Orbiters were only equipped to reach low Earth orbit (LEO), this is where the International Space Station (ISS), Hubble Space Telescope, and many other human made satellites reside. The Orbiters' main purpose was to construct the ISS in orbit and, service and deploy the Hubble Space Telescope. Unfortunately, the Orbiters would not be able to withstand the harsh space atmosphere that lies outside of low Earth orbit. Therefore, once the ISS was completed the program would be retired, but NASA would respond with the announcement of their new spacecraft, Orion, that could travel millions of miles away from Earth.

Orion is considered a Multi-Purpose Crew Vehicle (MPCV); visually this capsule looks very similar to that of the capsule used during the Apollo program. However, it is also very different; taking what they knew about the Apollo program and what they have learned through the Shuttle program NASA was able to design the Orion MPCV to be one of the safest spacecraft to date. With the added Launch Abort system, if there was ever a problem with the rocket on the Launchpad and the crew needed to escape quickly, this system would be able to shoot the capsule and the astronauts up and away from the rocket base. The capsule is also encased when stacked on the Space Launch System (SLS) rocket, unlike the Shuttle which was exposed to possible debris collision during launch. Also, the Orion capsule will be able to carry four astronauts unlike the Apollo capsule which could only carry three [Orion Quick]. At Kennedy Space Center, one of the main focus' is to design and develop the Ground Support Equipment (GSE) that will service both the new Orion MPCV capsule and the SLS rocket. Many of the old Shuttle processing and servicing facilities at Kennedy Space Center will be converted to accommodate the new Orion MPCV and SLS rocket.

I accepted an internship opportunity to work with the Spaceport Command and Control System as a remote display and application developer. During my time at Kennedy Space Center, I developed remote displays to be used in the Firing Rooms to help with the Ground Support Equipment for the Hypergolic Subsystem team for servicing the Orion MPCV. These displays will be controlling potentially hazardous fluids, so it is crucial that the displays be simple and as easy to read as possible for the console operator. I have also created a remote application that

will be used to channelize all the hypergolic remote displays. Once I have finished testing this application it will be ready to bring to NASA quality engineers for the necessary verification.

III. Hypergolic Subsystem

I am grateful to have been paired up with my subsystem mentor Joey Parkerson, he took the time out of his busy schedule to educate me on not only the software side of operations but also how the hypergolic subsystem of the Orion Multi-Purpose Crew Vehicle (MPCV) is designed and how hypergolics actually work. For this I am forever grateful, because this new type of knowledge and resources is not provided in a classroom or even public setting. Unfortunately, I cannot share the specifics on Orion's hypergolic subsystem but, he was able to show me schematics and provide me with documents to review that pertained to the specific processes that will be used to service the Orion MPCV and service module fuel and oxidizer tanks. On the other hand, for those that do not know what the hypergolic subsystem is, it is a system that combines a liquid fuel and a liquid oxidizer which when mixed produces combustion. This type of system does not require a source of ignition, the mixing of the both the oxidizer and the fuel causes the combustion. The hypergolic type of combustion is used for the reaction control thrusters and engine when the spacecraft is finally in space. Next time you watch a space movie or documentary look for the short bursts that help stabilize or move the spacecraft, as seen in Figure 1, this is the hypergolic subsystem in action firing the reaction control thrusters.

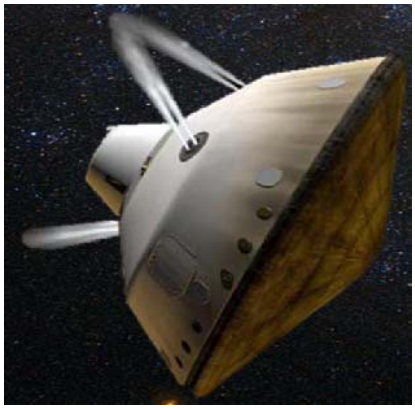

Figure 1: Example of the hypergolic subsystem, this is a computer rendered image of the capsule that housed the Mars Science Laboratory on its trip to Mars [Astronomy].

IV. Training

It took me and my other remote software development team members about three weeks to complete the required software training during my first internship experience. The two main software programs that we were trained on were NASA's abstraction layer of a programming language for remote application development and NASA's Display Editor for remote display development. We had both an in-class type of setting along with added self-guided documents and PowerPoints for both software programs. After completing our training classes and documents we were required to submit a skills demonstration for each software program. For the abstraction layer we need to demonstrate a set of code that used the abstraction layers calls, commands, and functions to successfully perform a task with the respective CUIs. For the Display Editor, we were required to create a display that had visual and logical order, and it also needed to include all four types of display symbols, which will be described in greater detail in

section six. Since, this is my second consecutive internship as a remote software developer I was able to quickly review the training materials, and pick up where I left off in the spring term.

V. Software Lifecycle

While going through training, one of the first required readings was on the software development lifecycle. I found that the software development life cycle can be a lengthy one, but it is also very important because it shapes how the system will be developed and implemented. This life cycle is broken up into five main parts, the 30% Design Review, the 60% Design Review, the 90% Design Review, Implementation, and Testing phase. In preparation for the 30% Design review is where many of the basic elements are determined, and the software side of the System Requirements and Design Specification (SRDS) document is created. At the 30%, only the expected amount of hardware and software interaction is included, along with the estimated complexity of both local and remote software need, and an approximate number of both the local and remote displays that need to be created. During preparation for the 60% Design Review the SRDS becomes better defined with what software components will need and their specifications. A major component included in this review is the Software Test Plan (STP). At this point, the STP is a general outline of test procedures and expectations, the detailed test steps will be added in the next stage. As a source for double checking that all major elements are included, the documents will go through peer review before reaching the 60% Design Review. The preparation for the 90% Design Review is similar to that of the 60%, but this time the updates to the documents will be more refined and finalized. Again, the documents will go through a peer review before reaching the formal 90% Design Review. After the documents have gone through all three phases of Design Review the team will enter the Implementation phase. This is where they will be able to start writing application software and creating displays in accordance to the SRDS that was approved through the Design Review phases [ILOA Training].

After I completed my required software training I was placed to work with the hypergolic sub-system team. When I entered the group they were finishing up their 90% Design Review and transitioning into the Implementation phase. This was a good time to enter because I was able to see firsthand how the team operated for both the 90% Design Review and the Implementation phase. The majority of my first internship experience was spent in the Implementation phase where I developed and edited many of the remote displays to be used in the Firing Room to service the Orion MPCV's hypergolic sub-system. For my second internship experience my focus was more on the Testing phase, were I developed and tested the display channelization remote application along with polishing the displays I made in the previous semester.

VI. Display Editor

NASA's version of human-machine interface development software was used to create the remote displays, where the editing workspace is called the Display Editor. The SRDS that is

created through the software lifecycle, as stated above, is the main document that outlines all the necessary components and a conceptual images design as a reference. While creating the display I worked very closely with the SRDS document. Not only did I use the document to help me design the displays, I also used this time to proof read the document for my hypergolic sub-system mentor.

The Display Editor has a very simplistic style of editing capabilities; the two main types of components were drawing components and symbols. The drawing components, as seen in Figure 2, were predominantly used as visual components with no physical functionality. These components were used mainly to separate and organize related groups of symbols and text together. More importantly, since the hypergolic sub-system works with both a fuel and an oxidizer, and the layout of each system is very similar so, the use of the color coded visual drawing components was very important in distinguishing between the two.

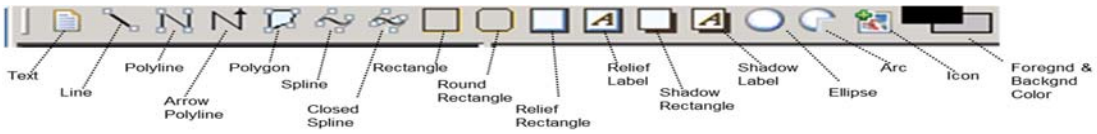

Figure 2: Drawing components tool bar in the Display Editor.

The symbol components are the heart and soul of the display, as seen in Figure 3 the symbols tool bar is comprised of four components: Text Measurements, Command Buttons, State Components, and Display Buttons. Three of the four of these symbols are tied to one or more Compact Unique Identifiers (CUI), which outlines specific limits, measurement types, commands, or runs a script file.

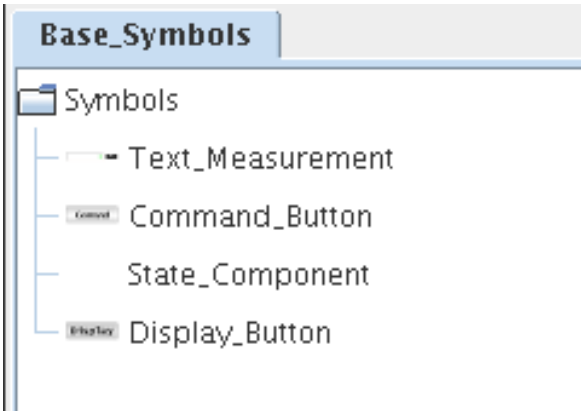

Figure 3: Symbols tool bar in the Display Editor

The text measurement symbol can display both numeric and enumerated measurements, such as; +1.00, 100.00, ON, OFF, OPEN, and CLOSE. Each text measurement symbol can only be tied to a single CUI.

Figure 4: Examples of text measurements.

Command buttons are used to issue commands such as turning something on or off, open or closed, or to primary or secondary. These command buttons can be tied to: an action, a status CUI, a script name, and at most 8 other arguments. The action argument is usually one word similar to what function you want the button to perform, such as start or stop. The status CUI argument is another CUI that controls the status of the command that button has performed, ensuring that the command has been sent successfully. The script name argument is the name of the script that the command button will find and then run through the code to complete the command. The other 8 arguments can be anything else that helps describe the button or what it is attached to, for example its hardware number or other specifics that are necessary to run the code in the script file.

Figure 5: Examples of command buttons.

The state components are used to show the state of a certain element or grouping of the system. These symbols are highly important for ease of use and eliminating confusion for the console operator in the Firing Room. Having these state components the console operator can know exactly what is happening throughout the system at a glance. To provide uniformity throughout all the displays a standard color scheme for the images has been set, this also decreases the likelihood of confusion. These symbols must be tied to a set of images along with a single CUI.

Figure 6: Examples of state components.

Display Buttons are the only symbols that are not tied to a CUI; however, they do tie all the displays together. These buttons allow you to open another display from one that you already have open. Many of the hypergolic displays are related to one another and console operator will need to have multiple displays open to analyze a specific part of the system. Unfortunately, both the display and command buttons look similar to one another. This forced me to add a color coded header bar to the top of each display which is where I laid out all the display buttons needed for that display to avoid command/display button confusion.

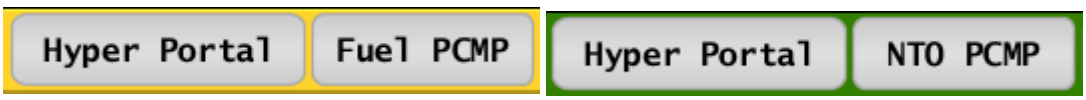

Figure 7: Examples of display buttons.

VII. Display Verification Sheets

Display verification sheets, or better known as DVS sheets, are used to help ensure all necessary components are included in the display and all the underlying information is correct. These sheets are generated from the actual graphical display file, therefore shows all the components that are currently included on the display. The DVS sheets look similar in format to a typical spreadsheet. They group all similar display components together for ease of comparison to the SRDS, for example all measurements together, all commands together, etc. Another, helpful feature that this sheet has is the unused images section. In this section is a list of all the images that are saved in the display's images folder but are not tied to any components on the display, therefore not needed in the images folder. This helps eliminate non-needed files and clutter within the file structure.

VIII. Previously Completed Tasks – Spring 2014

During the spring 2014 term I was also hired as a remote display and applications developer intern. After receiving and completing the required training I was assigned to work with the remote software for the hypergolic subsystem. During my spring term I predominately worked on creating, editing, and documenting the subsystems remote displays. I created and edited a total of 43 remote displays, and had all 43 built onto the testing sets in the Firing Room for integrated testing. To create the displays I used the SRDS to include all required components and the conceptual image provided as a layout reference. Before leaving for a short break in between terms I was able to do some preliminary integrated testing to see how the displays interacted with both the remote applications and the modeled hardware.

IX. Newly Completed Tasks – Summer 2014

Since, both this internship and my previous one in spring 2014 were working with the same subsystem and software components I was able to jump right back into where I left off in the spring. This helped me tremendously, considering the summer term is shorter than a typical fall or spring semester. I was able to briefly review most of the training requirements and introduction classes, and start getting to work on the updates that were made during my absence. Below are a few summaries of some tasks I completed during my summer internship experience.

My first task for the summer term was to go through a list of updates for each display that my mentor had made for me. This list contained items such as changing the names of certain components, adding new necessary components or deleting unnecessary components, and 1 new display. This display will only be loaded into the testing environment to help with the testing integration and verification process, and will never be loaded onto a live Firing Room set.

My next task was to generate a Display Verification Sheet (DVS), as stated above, for each display that will eventually be loaded onto the live sets and used for actual loading of hypergolic propellants. Once all sheets were generated I needed to compare the outputs in the DVS sheets against the Software Requirements and Design Specification (SRDS) document to ensure all required components were included in the display. At first, I began manually comparing the spreadsheets, and found that it was very time consuming. So, I used a macro enabled spreadsheet to compare the CUI column in the DVS sheet against the CUI column in the SRDS. This made the process a lot quicker and the differences between the documents more noticeable.

During integration testing my mentor found that some components looked different and some of the text ran off the command and display buttons. This discovery was not anticipated because, when opening the display file in the Display Editor the text fit inside the size of the button fine. Therefore, this new discovery forced me to edit and resize certain components in the Firing Room and pull them up on the working set to ensure that the corrections actually fixed the resizing issue. Another, issue found during integrated testing was that the default value for all the solenoid valves was 0.0, which is considered a double number. However, the code was looking for an integer default value. The resolution to this issue was to go through all the displays and update the 0.0 double default values to a default integer value of 0.

While I was absent between the spring term ending and the summer term starting, full time employee Bill Craig created 3 remote helium displays. These will control the filling of the gaseous helium tanks used by the hypergolic subsystem. I took his displays and edited them to look similar to the hypergolic subsystem's remote displays, for example the same type and size of text and a similar side and header bar. This was done to sustain uniformity throughout the displays for both hypergolics and gaseous helium used in the hypergolic propellant filling processes.

After creating all the displays in my previous internship experience I found myself with a decent sized list of updates that needed to be added to the display section of the SRDS. I requested a local copy of the most recent SRDS and have updated the tables in the display section accordingly.

My next major task was to create an Application Control Language (ACL) project that will be used to gain verification by NASA quality engineers. This ACL project was required to channelize all components included in the specific display, and if they have more than one state to show that all necessary states were also included. Another full time employee, Dean Breaux, wrote a very helpful tool called the Test Generator which eases the process for developing the actual ACL code. The Test Generator requires the user to create separate spreadsheet files for each display, which it will read to help generate the actual code structure. The spreadsheet needed to list every component and its necessary information, such as CUI, multiple states, and a brief name of the component. Creating these spreadsheets required me to set up how I wanted the test to run before filling in the information. For example, I did not was the quality engineer to

have to bounce around the display during verification which could cause confusion. Therefore, I established sub-test sets within the spreadsheet, so they could focus on a particular or similar group of components. The code will run through a single sub-test and then ask for acknowledgement that the user viewed the test, and it either passed, failed, or needs to be retried before moving onto the next sub-test. Once all the spreadsheets for each individual display were made, they were run through the Test Generator to generate the code that the ACL IDE could understand. From there, I needed to open and run the newly generated code in the ACL IDE environment and ensure that the code compiled successfully. After fixing some spacing and extended character length issues the code compiled successfully. Then, the next step was to promote all needed files into the software version control database and request that this new project be built onto the sets in the Firing Room testing environment. This new project will need to be tested and debugged before bringing it to the quality engineers for review and verification.

Another task that I have been assigned is to create 6 remote displays that will control flight hardware of Orion's onboard hypergolic subsystem. Three of which are focused on the command module, and the second 3 are focused on the service module. These displays require extra attention and planning as this is where the fuel and oxidizer come into contact with each other.

X. Beneficial Exposure

Currently, I am working to complete my Bachelors of Science in Aerospace Engineering at Embry-Riddle Aeronautical University. This internship opportunity has exposed me to many new skills and hands on software experience. I will be able to bring these newly acquired skills back with me to my university to help further my education, and more importantly in my future career. Not only have I acquired technical skills, I have also become more confident in my business communication and documentation skills. Being at NASA's Kennedy Space Center, I have had the opportunity to gain real world industry experience, and see many projects development, implementation, and testing cycles in real time.

Since I was in about third grade it has been my dream to one day work for NASA as an engineer and help create and develop spacecraft such as the Shuttle, and now the SLS-Orion. Being able to work with the hypergolic subsystem and learn more about how their system works has been a great honor. Creating the subsystem's remote displays and applications has showed me how both the hardware and software are linked together to support the end item, the Orion MPCV. It has been even more surreal to know that I have actually helped develop software that will be used in the Firing Rooms, relatively soon, to help service the actual space bound Orion capsule and service module.

XI. Conclusion

In conclusion, the remote displays that support the hypergolic sub-system need to be simple, easily readable, but still contain all necessary information to control the subsystem from the

remote location of the Firing Room. Being assigned as the sole remote display developer for the hypergolic subsystem team has provided me the experience to develop both my technical and leadership skills in a real workplace environment. I was able to show that I could balance the large responsibility along with completing tasks with due diligence. Taking on the task to write the display channelization remote application allowed me to enhance my decision making and planning skills. As this testing application did not have many guidelines and required me to create the layout and procedure of each test. Every display is different and required a different layout, therefore a pre-planning period was needed. In the end, I have found that I really like working with the hypergolic subsystem team and hope to acquire a full time position with an employer that deals with similar systems after completing my degree.

X. Appendix A – Acronyms and Abbreviations

Acronym/Abbreviation	Description
CM	Crew Module
CUI	Compact Unique Identifier
DC	Direct Current
GSE	Ground Support Equipment
HP GHe	High Pressure Gaseous Helium
ISS	International Space Station
LCC	Launch Control Center
LEO	Low Earth Orbit
MPCV	Multi-Purpose Crew Vehicle
MPPF	Multi-Purpose Processing Facility
SCCS	Spaceport Command and Control System
SLS	Space Launch System
SM	Service Module
SRDS	System Requirements and Design Specifications
STP	Software Test Plan
TCID	Test Control Identifier Document
VDC	Volts DC
VIP	Vehicle Interface Panel
VM	Virtual Machine
VPI	Valve Position Indicator

XI. Acknowledgements

I would first off like to thank NASA for the great opportunity I was given to experience how the aerospace industry works in real life, also thank you to all the NASA employees and contractors that have helped me complete my assigned tasks and taught me so many invaluable skills that I will continue to develop throughout my future career. I would especially like to thank my branch chief, Cheryle Mako for requesting interns and opening up this experience for college students like me. Also to my technical mentor Kurt Leucht for his guidance and patience

throughout the training process. Linda Crawford for all she has done to make us interns feel like part of the NASA family during our stay and for reviewing my final report before publication. Another thank you to Greg Clements for providing great opportunities for the division's interns to enhance leadership qualities, and also being a great role model for aspiring engineers. Also to, Bill Craig for taking the time out of his busy schedule to help me with any problems that I ran into along the way. Another thanks to, Dean Breaux for teaching me how to navigate through the computer operating system via a terminal window and for helping set up my test code project. Finally, I would like to greatly thank Joey Parkerson for his time, guidance, help with the software tools used to complete my tasks, and all of the knowledge he has shared with me about the hypergolic subsystem. He has gone above and beyond to ensure that I had a good understanding of how the subsystem works and how the displays I created become integrated in the final product.

XII. References

"NASA Facts: Orion Quick Facts", *NASA Lyndon B. Johnson Space Center*, Houston, Texas, 77068

NASA ILOA Core Team, "ILOA Training: The ILOA Software Development Lifecycle", NASA, September 6th, 2012.

Astronomy [website], URL http://www.astronomy.com//media/Images/News%20and%20Observing/News/2013/05/MSL_Orion.jpg [cited 07 April 2014].