Design and Development of Functionally Effective Human-Machine Interfaces for Firing Room Displays

Henry Cho
B.S.E. Aerospace Engineering
University of Michigan, Ann Arbor
NASA Kennedy Space Center
KSC FO: Fall Session
December 20, 2013

Design and Development of Functionally Effective Human-Machine Interfaces for Firing Room Displays

Henry Cho[*]
University of Michigan, Ann Arbor, MI 48105
Kennedy Space Center, FL 32899

This project involves creating software for support equipment used on the Space Launch System (SLS). The goal is to create applications and displays that will be used to remotely operate equipment from the firing room and will continue to support the SLS launch vehicle to the extent of its program. These displays include design practices that help to convey information effectively, such as minimizing distractions at normal operating state and displaying intentional distractions during a warning or alarm state. The general practice for creating an operator display is to reduce the detail of unimportant aspects of the display and promote focus on data and dynamic information. These practices include using minimalist design, using muted tones for background colors, using a standard font at a readable text size, displaying alarms visible for immediate attention, grouping data logically, and displaying data appropriately varying on the type of data. Users of these displays are more likely to stay focused on operating for longer periods by using design practices that reduce eye strain and fatigue. Effective operator displays will improve safety by reducing human errors during operation, which will help prevent catastrophic accidents. This report entails the details of my work on developing remote displays for the Hypergolic fuel servicing system. Before developing a prototype display, the design and requirements of the system are outlined and compiled into a document. Then each subsystem has schematic representations drawn that meet the specifications detailed in the document. The schematics are then used as the outline to create display representations of each subsystem. Each display is first tested individually. Then the displays are integrated with a prototype of the master system, and they are tested in a simulated environment then retested in the real environment. Extensive testing is important to ensure the displays function reliably as intended.

I. Introduction

As the end of the space shuttle program approached, NASA announced they would continue their efforts in space exploration beyond earth orbit. The space shuttles were intended to reach only low earth orbit where the International Space Station was assembled and where the Hubble Space Telescope was serviced for example. NASA announced the Orion Multi-Purpose Crew Vehicle, which is a space capsule similar in shape of the Apollo era space capsules but much larger and with modern technology. The Orion vehicle is designed to carry a crew of a maximum of six people, twice as many as the Apollo capsules. The Orion vehicle is much safer than the space shuttles because it is well protected during launch, whereas the space shuttles were exposed to debris collision during launch. It will launch on a redesigned launch vehicle called the Space Launch System, based on the canceled Ares launch vehicle. The operations at Kennedy Space Center needed to develop ground support equipment to service these new vehicles. I accepted an internship and assigned to work on Integrated Launch Operation Applications as part of Control and Data Systems at KSC. During my internship, I developed display software used to interface the ground servicing equipment with the launch vehicle and Orion vehicle. An example of one of these displays is shown on Fig. 1.

[*] Display Developer for Firing Room Applications, Integrated Launch Operations Applications, Kennedy Space Center, University of Michigan: Ann Arbor.

II. Requirements and Design

A. Schematics

The schematics of the hardware system of the Hypergolic fueling system were designed by fluids engineers from another department. These schematics are given to software developers to use as a basis for designing the software used to control the physical devices in the field. These schematics were used as the direct basis for designing and developing the remote displays used to control the Hypergolic fueling system. They were adjusted for the displays to avoid crossing fueling lines to reduce confusion of the direction of flow.

B. Software Requirements and Design Specification

The design phase of a software development lifecycle is the first phase but the most time-consuming phase because it lays the framework for what is required and how to develop it. The design phase is divided into three sub-phases: 30% design, 60% design, and 90% design.[1] During the 30% phase, only the initial concepts and estimates are outlined in the SRDS document. This phase includes estimates of the hardware required, size and complexity of the developing software, and the estimated number of displays. During the 60% phase, the details of the software are better defined. A software test plan (STP) is developed in the 60% phase which is a brief outline of the expected results. By the end of the 90% design review, the software design and requirements are refined and detailed. Around the end of the 90% phase, the team is ready to begin development and implementation of the software designed.

III. Development and Implementation

A. Minimalism

Minimalism is a design style that reduces the number of non-essential features to produce a clean feel commonly used to convey information effectively. Minimalism grew in popularity during the computer era and became widespread more recently when displays and printers advanced to output at high resolutions to be able to produce curved lines with smooth definition, making individual pixels and dots less apparent. This allowed fonts to be displayed more defined curves, where each character appear as one object, not as a collection of pixels used to approximate each character.

The font Lucida Sans Typewriter that appears in my displays is poorly rendered. I do not know with certainty whether it is the virtual machine fonts, Linux fonts, or the Java-developed Display Editor that which limits the quality of the rendering. I speculate that the virtual machine limits the quality of graphical render, most likely to improve performance. It was decided Lucida Sans Typewriter was the font required to use for text on all displays because it is a standard font package included on all computers supported by NASA. This font is a sans-serif font, as indicated in the font name. *Sans*, meaning "without," indicates a font that does not have *serifs*, the protruding features at the ends of a character such as in Times New Roman.

Lucida Sans Typewriter is a good font to design displays because it also clarifies the ambiguity of most similarly shaped characters as shown on Fig. 1. The two most similar characters are the uppercase o and the number 0. The uppercase o has a slightly larger width than the number 0 but there is no obvious indication to differentiate between the two characters. Especially the characters uppercase i and lowercase L are often confused in Helvetica-based fonts. An example of this in Arial, which is very similar to Helvetica, is shown in Fig. 2. These two characters look almost identical but they are not. The uppercase i is slightly thicker than the lowercase L. Lucida Sans Typewriter clears this confusion by adding protruding features on these characters.

o	O	0
Lowercase o	Uppercase o	Number 0
l	7	1
Lowercase L	Number 7	Number 1
l	i	I
Lowercase L	Lowercase i	Uppercase i
z	Z	2
Lowercase z	Uppercase z	Number 2
s	S	5
Lowercase s	Uppercase s	Number 5

i	l	I	1	7
Lowercase i	Lowercase L	Uppercase i	Number 1	Number 7

Figure 1. Comparison of Similarly Shaped Characters in Lucida Sans Typewriter

Figure 2. Comparison of Vertical Shaped Characters in Arial

B. Colors

Different colors indicate which commodity chemical flow through an active component and helps distinguish from other commodities. The colors with a specific color name are taken from the X11 color scheme.[2] X11 is a windowing graphical interface for UNIX-like operating systems and is packaged with a list of RGB color values with specific names. At least one name was assigned to each specific RGB color value. The X11 colors became the basis for web colors.

The commodities without color names are off-colors, not based from the X11 color scheme. Two different shades of yellow were used to differentiate between Hydrazine and Monomethylhydrazine. The last color on Table 1 is not assigned to a specific commodity but is used to color passive components used throughout all displays. A list of these passive components is shown on Fig. 3.

Table 1. Common Commodity Colors

Commodity	Color Name	R	G	B	Hexadecimal	Example
Air	Light Steel Blue	176	196	222	#B0C4DE	🟦
Ammonia	Light Pink	255	182	193	#FFB6C1	🟥
Freon	Aquamarine	127	255	212	#7FFFD4	🟩
Helium	Tan	210	180	140	#D2B48C	🟫
Hydraulic Fluid	Salmon	250	128	114	#FA8072	🟧
Hydrazine	–	255	236	117	#FFEC75	🟨
Monomethylhydrazine	–	255	212	42	#FFD42A	🟨
Hydrogen	Khaki	240	230	140	#F0E68C	🟨
Nitrogen	Pale Turquoise	175	238	238	#AFEEEE	🟦
Nitrogen-Tetroxide	–	51	128	0	#338000	🟩
Oxygen	Pale Green	152	251	152	#98FB98	🟩
Water	Medium Blue	0	0	204	#0000CD	🟦
–	–	96	74	123	#604A7B	🟪

Most of my displays have a solid gray color (RGB 204, 204, 204) as a background, intended to reduce eye strain compared to traditional bright white display backgrounds. My displays that do not have the gray background are my integrated displays, which are displays that combine smaller displays to better represent the whole system. The integrated displays have background colors that match the header color of its corresponding display. Refer to Appendix D for integrated displays.

The color on the header indicates which commodity gets delivered through that part of the system. For example, the color on the header is RGB 255, 212, 42, which represents Monomethyhydrazine being delivered. The header colors on Hypergolic displays include a light-yellow color and a gold color, representing the fuels Hydrazine and Monomethylhydrazine respectively and a green representing the oxidizer Nitrogen Tetroxide N_2O_4.

For Hypergolic displays, the active components are grouped in rectangles with a fill color corresponding to the commodity that flows through it. An example of an active component, a solenoid valve, is shown in Fig. 4, which is grouped in a pale turquoise indicating gaseous Nitrogen flows through it. The vehicle interface panels have a solenoid valve that mixes the flow of two gases, Helium and Nitrogen. These components are grouped using a rectangle with a color gradient from tan to pale turquoise to represent the mixture. Refer to Appendix C for vehicle interface panels.

C. Components

The displays are comprised of passive components and active components. Passive components include background shapes with fill colors, lines and arrows directing the path of the flow, and text box labels. These components help group certain information together and help flow information in a certain direction. Passive components specific to the Hypergolic fueling system are shown in Fig. 3. These symbols represent physical devices out in the field, where personnel would have to be physically present in the field to operate, such as a manual valve.

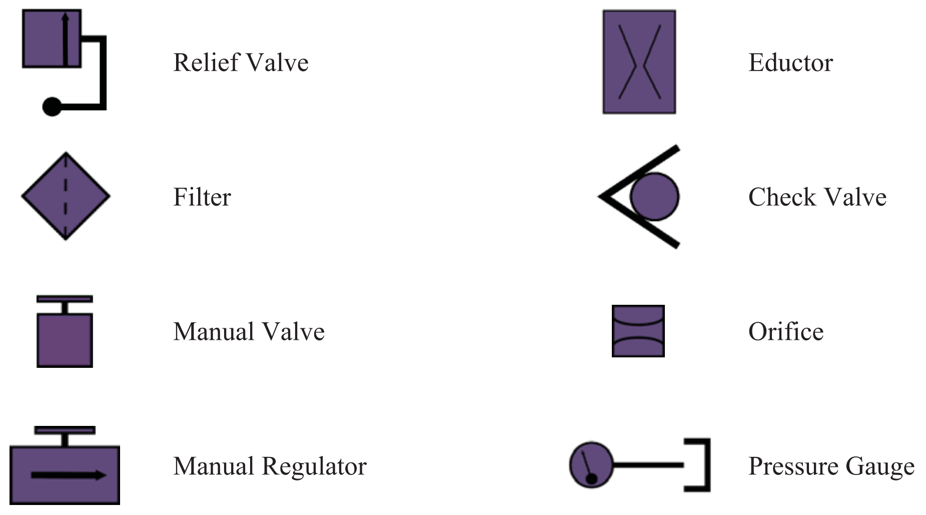

Figure 3. Display Symbols of Passive Components

Active components of a display include the command buttons, measurements, and state indicators. Examples of active components are shown in the solenoid valve below in Fig. 4, which include command buttons and state indicators. These active components use a name called a Compact Unique Identifier (CUI), which is a set of alpha-numeric characters, effectively a global system variable that carries a command or measurement signal. There are naming convention guidelines when creating CUIs that identifies its project, system, and subsystem.[3] An example of a CUI is shown on Fig. 5. The fifteenth character represents the data type the CUI carries listed on Table 2.[4] A CUI is inserted into the properties of each component to uniquely identify the command or data measurement that passes through the system. Command buttons have a command type CUI. Both alarm indicators and valve position indicators have enumeration type CUIs, which is a data type with choices preset in the system. For example on Fig. 6, alarm indicators can display one of four possible states: Default, On, Off, and Error. The images are used to visually represent the data, such as for valve position indicator states in Fig. 7 with six possible states. The centerline on the Closed image in Fig. 7 is intended to represent a blocked path, perpendicular to a horizontally represented fueling path. Use of either the vertical or horizontal centerline depends on whether the fueling path is horizontally or vertically represented to correctly indicate a closed fueling path.

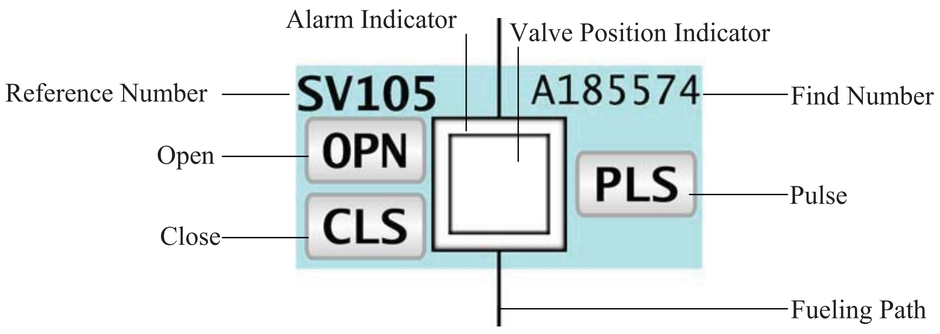

Figure 4. Solenoid Valve Comprised of Passive and Active Components

Reference Number	Numbers used to easily identify active components, e.g. SV105
Find Number	Numbers from the schematic used to uniquely identify every part, e.g. A185574
Open	Command button to open valve
Close	Command button to close valve
Pulse	Command button to open valve for a set valve between 0.0 s to 5.0 s, then closes
Alarm Indicator	Displays an alarm state of *Default*, *On*, *Off*, or *Error*

Valve Position Indicator	Displays a valve state of *Default*, *Open*, *Closed*, *OPPRI*, *OPSEC*, or *Error*

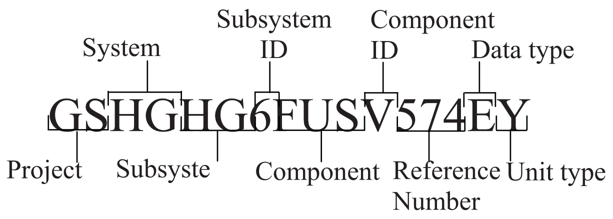

Figure 5. Compact Unique Identifier for a Valve Position Indicator for the Hypergolics System with an Enumerated Data

Table 2. Data Types of Compact Unique Identifiers

Fifteenth Character	**Data Type**
–A..	Array
–B..	Boolean
–D..	Derived
–E..	Enumerated
–F..	Floating point
–G..	Group
–H..	Hexadecimal
–I..	Integer
–K..	Command
–M..	Modifiable command
–P..	Predefined command
–R..	Raw command
–S..	Command structure
–T..	Text

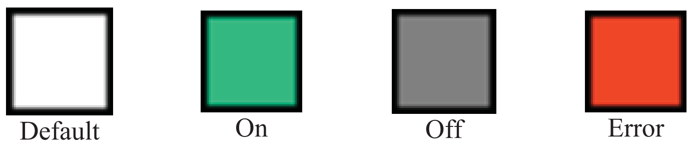

Figure 6. Alarm Indicator States

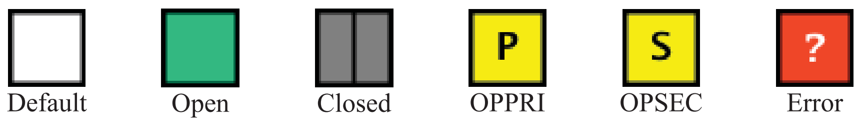

Figure 7. Valve Position Indicator States

IV. Future Work and Improvements

A. Displays

1. All the display navigation buttons, which link to other displays, need to be corrected to their corresponding file names.

2. All the pressure and temperature transducers active components, represented by the prefixes *PX-* and *TX-* on the reference names, need to be added into the displays.

3. All the sidebars need to be updated with the correct components.

4. All the displays need to be tested to ensure that their active components correctly control the intended device.

B. Display Editor

As I was working with the Display Editor, I discovered that there are basic capabilities missing, those of which are often found in common graphical editing software. I also found bugs and imagined potentially new features while working with the Display Editor, which should be corrected and added in future versions of the Display Editor.

1. Copy and Paste Problem

When I select an object then copy it and I select another object without copying it, when I paste the Display Editor will paste the second object I selected. Copy and paste does not function as a user would expect because the Display Editor should retain the copied object. The problem is especially apparent when copying and pasting across two displays. The problem should be fixed such that the copied object is retained as the correct object to be pasted.

2. Misleading Background and Foreground Color Options

The options to change the background and foreground colors are misleading shown in Fig. 8. Initially, my supervisor and I thought the option to change the background color was to apply a background color on an entire display. We thought it was a non-functional option until I discovered changing the background changes the fill color of an object. Shortly after, I discovered the foreground color is used to change stroke color of an object. The naming convention should be changed from "Background color" to "Fill color" and "Foreground color" to "Stroke color"

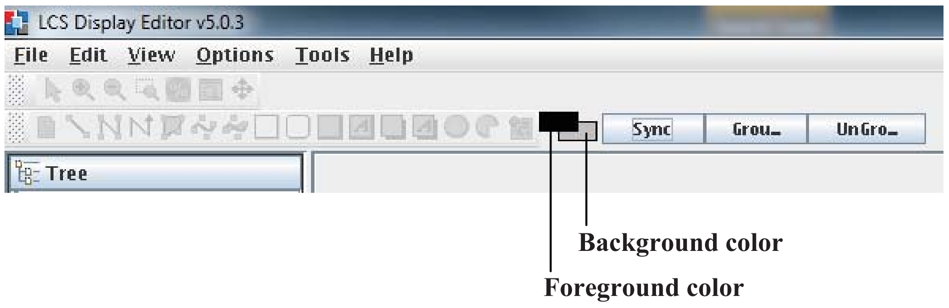

Figure 8. Background Color and Foreground Color in LCS Display Editor

3. Rotate and Flip Options

The option to rotate or flip an object does not exist in the Display Editor. There should be options to rotate an object 90º, 180º, 270º, and at an arbitrary degree set by the user. There should also be options to flip an object horizontally and vertically. Rotating and flipping achieve different results, which is apparent when applied to irregularly shaped objects.

4. Unintended Alignment of Group Objects

When grouping a set of objects, the Display Editor does not create a new coordinate for the center or new height and width dimensions of the group. Then, when aligning grouped objects with other objects, each separate object within the group uses its own dimensions to align. For example, when aligning the vertical centers with a group of objects, each separate object within will align its center coordinate with the other center coordinates as shown on Fig. 9 below. When grouping a set of objects, a new center coordinate should be created as well as new width and height dimensions such that the group should function as a single object.

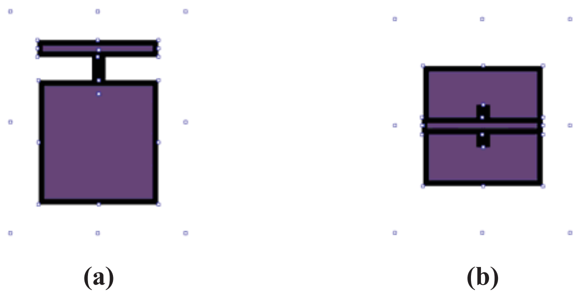

Figure 9. (a) Grouped Manual Valve Constructed from Two Rectangle Objects and One Thick-Stroke Line, (b) Grouped Manual Valve Aligned at Vertical Center

5. False-Movable Object Moves During Alignment

When an object's *movable* property is set to false, the Display Editor only restricts the user's direct control of an object's location. The object is not allowed to move by clicking and dragging the object. However, when aligning the false-movable object with another object, it will still change its location coordinates to satisfy the alignment. This makes it difficult to precisely align objects on the header and sidebar. Setting the *editable* property to false also continues to allow its coordinates to change during alignment. This should be fixed such that the false-movable object remains unable to move and movable objects would align to its center.

6. Unconventional Controls of Zoom Magnification

The controls for zoom magnification are inverted compared to most conventions, such as those in Microsoft Word and modern web browsers. Currently, the Display Editor zooms in with *ctrl + scroll down* and zooms out with *ctrl + scroll up*. This is inverted from most magnification conventions. Normally to increase magnification, the user would use *ctrl + scroll up* and to decrease magnification use *ctrl + scroll down*. I think the controls should be corrected to meet this convention. Also, a new feature should be implemented that zooms towards and from the coordinates of the mouse cursor when zooming in and out. Currently, it zooms in and out at the center of the display area, having to adjust with the scroll bars.

7. Shortcut Keys

Shortcut keys should be mapped to commonly used actions and commonly accessed options such as setting alignment, accessing the prompts to insert the CUI and image arguments, and toggling the grid with different grid spacing settings. Usage of the mouse was an improvement for ease of learning how to use computers through graphical interfaces. However, users should return to using keyboard shortcuts and even proceed towards scripted automation if possible. The repetitive motions from using only the mouse to access a tool and then return to the location to apply the tool is inefficient, when the other hand on the keyboard is immediately available to push a button to access the same tool, all the while the mouse cursor is at the location to use it. Keyboard shortcuts are necessary to optimally use design applications. This is apparent in Adobe creative applications and computer-aided engineering software.

V. Conclusion

In conclusion, human-machine interfaces that will be used to control potentially hazardous devices should be designed to convey information effectively to reduce the likelihood of human error. For this reason, the displays for the firing rooms are designed with a minimalistic style that is intended to be clean and intuitive while still conveying the necessary information to operate the interface. A functionally effective interface also includes fonts that differentiate between each character clearly and certain colors to focus on certain groups of information as well as to differentiate one group from another group. Future improvements to these displays will likely continue to use a minimalistic style but with the focus of information readjusted, such as increasing the contrast between background elements and dynamic information components.

Appendix A. Acronyms and Abbreviations

Acronym/Abbreviation	Description
CAE	Computer-Aided Engineering
CM	Crew Module
CUI	Compact Unique Identifier
DE	Display Editor
GHe	Gaseous Helium
HP GHe	High Pressure Gaseous Helium
Hz	Hydrazine
LCS	Launch Control System
MMH	Monomethylhydrazine
MPPF	Multi-Purpose Processing Facility
PCMP	Propellant Container Manifold Panel
PF	Post-Flight
PPP	Pressurization and Purge Panel
RGB	Red-Green-Blue color model
SLS	Space Launch System launch vehicle
SM	Service Module
SRDS	Software Requirements and Design Specification document
STP	Software Test Plan
SV	Solenoid Valve
VIP	Vehicle Interface Panel

Acknowledgments

Henry Cho thanks NASA Education for their internship program, Cheryle Mako for requesting interns through their program, Kurt Leucht for his guidance during my internship, Joey Parkerson for his guidance on the software I helped developed, and Gregory Clements for reviewing this report before publication.

References

[1]Leucht, K., "ILOA Software Development Lifecycle," 2013, pp. 10-16 [cited 21 November 2013].

[2]dawes, "rgb.txt," *XFree86 CVS Repository*, Rev. 1.1,
URL: http://cvsweb.xfree86.org/cvsweb/xc/programs/rgb/rgb.txt?rev=1.1&content-type=text/vnd.viewcvs-markup
[cited 21 November 2013].

[3]Leucht, K., "ILOA Implementation Standards," 2013, pp. 21-22 [cited 3 December 2013].

[4] "CUI Character Lists," *Integrated Launch Operations Application Software Wiki Home Page*, August 2009, URL: https://nasa-ice.nasa.gov/confluence/display/GOLCSILOA/LCS+Integrated+Launch+Operations+Application+Software
[cited 4 December 2013].