A Common Command Interface for Interactive FACE Units of Conformance (UoC)

Implementing a Menu System in the UH-60 Crew Mission System (CMS)

Air Force FACE™ TIM Paper by:

Steven Price & Chris Edwards
AMRDEC/ASIF

February, 2017
A Common Command Interface for Interactive UoCs
# Table of Contents

**Executive Summary** ................................................................. 5

**Description of Problem** .......................................................... 6
  The use of ARINC-661 to Provide Partitioning of User Interfaces .......... 6
  The Addition of UoC’s that Accept Commands/Events from User Interfaces 6
  Positioning of Buttons ................................................................... 6
  User Application Reuse Across Differing Systems .............................. 6

**Background** ............................................................................. 8
  Design Assurance Levels (DAL) and Partitioning ............................... 8
  Cockpit Display System (CDS) and User Applications (UAs) ............... 9
  Style .......................................................................................... 10
  A Menu System ............................................................................ 10

**The CMS Menu System** ............................................................ 12
  Menu Widgets ............................................................................... 12
  Physical Buttons .......................................................................... 12
  Command / Event and Status Interface ............................................. 13
  Menu Configuration File .................................................................. 15

**Menu System Interface** ............................................................ 16
  Menu System Event Message ......................................................... 16
  Application State Data Message ..................................................... 16

**Conclusion** ............................................................................... 17

**References** ............................................................................... 18

**About the Author(s)** ................................................................. 19

**About The Open Group FACE™ Consortium** .............................. 20
A Common Command Interface for Interactive UoCs

About The Open Group..............................................................20
Executive Summary

The FACE Technical Standard provides a framework for the development of independent capabilities and systems that can support them. The UH-60 Crew Mission System (CMS) has been developed as an architecture to provide new capabilities to the field in the shortest time frame possible. One of the primary ways of reducing the time to field is to develop components that can be configured without the re-compile and related retesting of these components.

The FACE Technical Standard recommends ARINC-661 as a means of presenting information from multiple applications competing for screen real-estate. ARINC-661 was developed by the Airline Electronic Engineering Committee (AEEC) and published by ARINC in 2001 for use in new aircraft designs. This standard provides a means to control access to the display real-estate when multiple independent applications may attempt to access the same locations. The controlling application, the Cockpit Display System (CDS) partitions the screen using a “widget” interface. One of the applications is a Window Manager that dictates which application’s widgets are displayed given the states and modes of the current display.

When a system is being developed to support applications critical to aircraft operations, these applications should be partitioned away from the applications that are not critical to aircraft operations. This allows the non-critical applications to be developed to a lower level of rigor than the critical applications. An ARINC-661 CDS can be developed as a “critical” application and can be used to partition screen real-estate to “non-critical” applications running in another partition. In order for this to work, the Window Manager must also be developed to the critical level of rigor.

To meet the CMS goal of faster fielding, which could be accomplished by reducing the number of applications that need to be recompiled, the CMS team developed a configurable Window Manager and Menu System. This application explores a new interface that provides a means of sending user commands to other applications, whether those applications are using ARINC-661, or are simply supporting this basic command interface. This menu-system interface, if used within portable applications, promotes re-use throughout FACE Systems also using this interface.

This paper addresses the need for this interface, the flexibility of the interface to support varying hardware, and the advantages the interface brings to integration of applications utilizing ARINC-661.
A Common Command Interface for Interactive UoCs

Description of Problem

Using of ARINC-661 to Provide Partitioning of User Interfaces

There is a desire within CMS to reduce the modifications and re-testing of applications. Ideally, the applications requiring the highest level of rigor in development would not be affected by the addition of applications developed to a lower level of rigor. This can allow the deployment of simple applications without expensive rework of higher rigor development tasks. Breaking this dependency also aligns with the re-use and cost savings goals of FACE.

The Cockpit Display System (CDS) is the only application that can render to the screen. Its configuration files specify the widgets that an application can draw, and the boundaries for those widgets. User Applications are restricted to sending commands related to the widgets in their related configuration file. This allows the ARINC-661 CDS to receive communications from partitions developed to less rigor. This makes the CDS the gate-keeper for the display device, allowing applications developed with less rigor to share the screen with applications supporting critical operations for the aircraft. The Window Manager, as the application that controls what widgets are currently allowed to draw on the screen, also must be developed to the highest level of rigor of any application that renders to the screen.

Adding UoC’s that Accept Commands/Events from User Interfaces

In many cases additional capabilities could be added to a system through UoCs that only need simple commands (such as on/off) and do not display more than the state of these commands. In these cases the UoC could provide that information to a generic User Interface application that controls settings throughout the system. This application, however, would need modification as capabilities requiring its use are added. The CMS architecture would benefit from a User Interface Application that could handle additional capabilities through configuration changes rather than recompiling.

Positioning of Buttons

The development of a User Application (UA) involves the placement of widgets in relative locations within the context of the information displayed by that UA. Generally, buttons related to soft-keys are developed to coincide with the location of bezel buttons on the individual display that the UA is deployed on. This design paradigm results in application interfaces that are closely tied to the original equipment. The CMS architecture has been developed to easily replace its selected hardware. UoC’s developed for CMS should not have dependencies on the bezel locations of the currently selected hardware.

Reusing User Applications Across Differing Systems

Ideally, the CMS system will adopt new capabilities from the FACE Repository. These UoCs likely would not have been developed with the CMS system in mind. They were more apt to be developed with different style requirements, and to hardware with different bezels or user interface devices. ARINC-661 has an answer for much of this problem. The CDS defines the style for buttons and other widgets, but the selection of the type of widget and its related symbology is still defined by the UA. Consistency of the look and feel from UA to UA affects ease of use and is a key element in user interface standards (such as MIL-STD-
A Common Command Interface for Interactive UoCs

1472G). The style and location of “menu” like items must be determined by the integrator. Integration of user interfaces requires knowledge of the hardware and the mission of the particular aircraft.
Background

**Design Assurance Levels (DAL) and Partitioning**

Within the domain of flight safety, software is classified into Design Assurance Levels (DAL) based on the impacts the software can have to the aircraft operations if it were to fail. These concepts are expressed in RTCA DO-178C and referenced within the DoD Joint Software Systems Safety Engineering Handbook. Within these guidelines a system is expressed in terms of these assurance levels.

Partitioning is an architectural concept used within DO-178C to define a way to isolate potential faults. Application software that is not as critical to the entire system can be executed in an isolated manner that restricts its impacts on the higher criticality aspects of the host system.

An operating system capable of providing software partitions is one of the fundamental aspects of the FACE Operating System Safety Profiles. The ability to deploy UoCs that can be separated into different partitions
A Common Command Interface for Interactive UoCs

can greatly reduce the amount of proof needed to show these additional UoCs can’t effect the operations of other UoCs already deployed into other partitions.

Cockpit Display System (CDS) and User Applications (UAs) in a FACE Implementation

The CDS is a UoC within the Graphics Services PSSS Sub-Segment that is responsible for actually sending information to the display. Within a properly partitioned system, this UoC is the only software in the system that has the ability to use the display device.

The CDS uses configuration files that include the graphical description of everything the connected User Applications are allowed to draw. This configuration is expressed in terms of widgets. The CDS receives ARINC-661 messages from the User Applications (UA) instructing change to the parameters of widgets previously defined.

In ARINC 661 Appendix H describes the need for a UA that controls which applications are displayed and in what priority. When a single display has multiple applications that use a large portion of the display real-estate, the window manager defines states in which each application has control of some portion of the display. This UA is called the Window Manager, and one of the layers it controls is the Super Layer. The Super Layer defines what application widgets are visible within each “page” of the system.

The ARINC 661 Super Layer contains one or more personalities (or Pages). The Pages are mutually exclusive and can be selected at runtime. Each page can have one or more windows where application Layers can be displayed (confined by the window). Each window contains one or more layers each of which can be controlled by a single UA. Each layer contains the widgets a given UA can modify.

Windows can prevent errant applications from drawing outside of their defined space, and pages can prevent them from drawing at all when it is inappropriate. Protections can be built into the CDS and Window

---

1 One of the widgets supported by ARINC-661 is an ExternalSourceWidget which can allow OpenGL, Video, or other rendering to be passed to the CDS for display. This rendering is confined to the boundaries established by the ExternalSourceWidget, and may have other widgets placed over it. The CDS is responsible for placing the rendering on the display within these constraints.
Manager Configuration files to prevent widgets from exceeding specified boundaries, or to prevent their visibility if the Window Manager UA determines those widgets should not be shown. While the Window Manager can hide widgets from any UA, the widgets from a UA can only be manipulated by that single UA.

Within the constraints of windows and pages a UA can be added to a system in a new partition, that UA can be given a connection to the CDS, and the CDS configuration files altered to support that UA, it can not interfere with the function of other applications already on the system. The only proof needed is an examination of the configuration files for the CDS, specifically those of the Window Manager UA. These two applications, the CDS and the Window Manager, would have to be placed in a partition where the software is qualified to the highest level of any application that would display things through the CDS.

The CMS team plans to use these rules to define control of the screen real-estate to a set of two UoCs, the CDS and the “Window Manager / Menu System”. The intention is to develop these applications once (or as infrequently as possible), and use only changes in their configuration files to add capability to the CMS system.

The CDS also generates events that are sent via ARINC 661 messages, to the UAs to indicate user interaction. UAs generally use this feedback to provide control to the user over the UA.

Style

Within ARINC 661, the CDS controls the “style” of the display. UA’s may define buttons, panels, and lists, but the CDS determines what those widgets look like. This behavior allows a CDS on a new system to use an application developed for a different system to look like it was developed for the new system.

“StyleSet allows the UA to select from a predefined set of graphical characteristics to be applied to a widget. This serves two purposes. First, many graphical capabilities (color depth, halo, fill styles, line weights/patterns, blinking, transparency, fonts, character highlighting, kerning, rotation, etc.) are inherently a function of CDS architecture.

Second, the application of these characteristics is usually intended by the aircraft OEM to be consistent across all UAs for common state conditions. Indexing among predefined styles supports this goal. Common state conditions can be defined as conditions that impact more than one user application in the same way (e.g., Alert, Caution).”

- Definition of a StyleSet, ARINC 661 supplement 5, section 3.1.3.3

ARINC 661 also includes the concept of StyleSets. The style(s) for many widgets are defined in the CDS and an integer is used to indicate the choice of styles. Each individual UA defines which widgets, from the set of defined widgets in ARINC 661, it uses, and their relative locations. The CDS determines what the widgets look like, and the relative position of the set of a UA’s widgets with respect to other UAs.

A Menu System

This concept of the CDS owning the style is a practical way to allow applications to adopt the look-and-feel of the new host system. When exploring how to add new capabilities to the CMS system, the CMS team realized that the selection of “pages” and options that are supported by those pages must also share the look and feel of the end system. Not only that, but the selection of the widget type used is system dependant, and the location of options within these menus must be in the control of the System Integrator.
A Common Command Interface for Interactive UoCs

The System Integrator will be able to respond to requests by the end-user on the priorities each UA has within the specific deployment of the end system. Crew selection for which page to display and what page options are available has more to do with the integration of all UAs to suit the needs of the end user than it does with the selections made by the development of the original system.

A UA developer may allow multiple options for the display of a map, but the users on one system may only be interested in two or three of those options. The user interface dedicated to those undesired options could be better served supporting other options the end user wants.

This leads to another issue, if the menu is displayed through a single UA, how does that UA send commands to the other UAs and how does the Menu UA know the status of the commands to properly reflect availability?

“The DataConnector is similar to a regular Connector widget, but in addition it provides the ability for an array of parametric information to be passed to the Application that owns the referenced (child) layer.”

“The parametric data that is sent does not need to be interpreted by the CDS. The meaning of the parametric data only needs to be known by the designer creating the layer in which the DataConnector resides and the designer of the layer to which the connector points.”

ARINC 661 supplement 6 is offering a DataConnector as a means to send data from one UA to another UA. Unfortunately the data is not defined in the standard, and both UAs must be developed to an agreed upon set of parameters and their meanings. This is the kind of interface that the FACE Technical Standard is trying to avoid.

That opportunity for inconsistency could result in a fair amount of effort and rework to integrate new capabilities into a system, or porting from one platform to another. Having well-defined command and status messages used within the FACE Data Architecture will enhance the portability of applications with less impact to the target systems.

Within the CMS environment we may also have UoCs that have not provided an A661 interface but do provide commands and status. A good example might be a PSSS Device Service that provides some controls that are not included in any UA. The menu system needs to provide a consistent look and feel for sending commands to these applications as well.

Through exploration of ARINC-661, it is not reasonable for each application to draw its own menu options, nor is it reasonable for the communication of a general-purpose interface to direct commands to other UAs through ARINC 661 and still meet the intent of the FACE Data Architecture.
A Common Command Interface for Interactive UoCs

The CMS Menu System

In order to meet its goal of rapid deployment of new capabilities, the CMS system requires a system for rendering to the screen that supports the separation of concerns partitioning can provide. The system should be implemented in such a way as to allow the rendering system to be compiled once and configured to support the applications that are added through changes in configuration files only.

The CMS system also needs a way of configuring the UA that drives a Menu System by the same rules. Since the Window Manager is intended to be the owner of the currently presented page, and the selection of the current page is a menu function, it makes sense to have this UA also serve the Menu function.

This menu system must also communicate events to the other UAs as well as obtain the application state data to properly display a widget to indicate what interactions are available. CMS must define a data modeled interface for receiving status and sending commands.

Menu Widgets

The Menu System will need a layer of widgets that can represent all commands that can be configured. These can be text only, or be text and graphics. The set of widgets, their locations, and the icons used for graphics can all be stored in the Definition File (DF) supplied to the CDS as part of the menu configuration.

An effective menu system would use consistent widgets with their relative positions defined by the menu system, potentially collocated with physical input device, like bezel keys.

The widgets and icons in the DF are identified with ID numbers. These IDs are passed to the CDS when the menu system addresses these widgets or selects these icons.

This can allow the Menu System to be compiled with a maximum number of widgets, and have a configuration file that supports less than this maximum. The menu system logic doesn’t care about the locations of the widgets, just their IDs. This allows the DF to define new locations for menu buttons without the need to update the menu system code.

Physical Buttons

The Menu System will need to take inputs from the ARINC-661 widgets, some of which are co-located with physical bezel buttons (softkeys), as well as from the bezel buttons themselves, some of which are not represented by ARINC-661 widgets (hardkey only).

Figure 3- CMS Display Keys

Buttons with Softkeys

Buttons without Softkeys
Every application that accepts input from the user can view these user interactions as an event. These events can be realized as a single event with no associated data. For example “Next Page”, “Hide Symbols”, “Show Symbols”, “Full Page”, “Half Page”, “Increase”, “Decrease”, etc. When the interface is more complicated than these simple events, usually the interface is not associated with a menu.

For each of these items, the application may have states when the event would not be accepted, when the associated function is active, or even when the associated function is encouraged.

The Menu displays buttons or other widgets to allow user selection of these events. The Menu can use the state data published by the applications to determine how to render the interface.
A Common Command Interface for Interactive UoCs

For the CMS system, the possible rendering states include those listed in Table 1. The configuration of the menu can map the application conditions to whichever state the integrator chooses. In some instances, the unavailability of a command will lead to a “disabled” state, where in others it may lead to an “invisible” state.

Table 1: Possible states that can be displayed

<table>
<thead>
<tr>
<th>State</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Inactive</td>
</tr>
<tr>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>Selected</td>
<td>Not-Selected</td>
</tr>
<tr>
<td>Visible</td>
<td>Invisible</td>
</tr>
<tr>
<td>Emphasized</td>
<td>Nonemphasized</td>
</tr>
</tbody>
</table>

The support all of these states is achieved by defining logical conditions by which these states would be set true or false based on the application state data being sent to the Menu System, and configuration. Multiple states can be set with any particular set of configured conditions.

The interface back to the application need only include the command ID.

An addition of a UA ID to each of these messages can provide some level of assurance the system configuration and the TSS routing are all properly established.
A Common Command Interface for Interactive UoCs

Menu System Configuration File

The Menu System configuration file is a file read by the Menu System to correlate the widgets in the DF to the commands provided by the applications, to the pages used within the Super Layer.

In order to support the interfaces in a generic manner, this configuration file must include:

- the IDs for all widgets in the DF that will be used as buttons or labels for buttons
- the list of applications (UoCs) connected to the menu system (related to a generic ID)
- for each application, a list of Menu Events identified by the ID that the application will recognize
- for each application, a list of Application States identified by the ID that the application will send
- the list of “pages”, as expressed within the super layer with the associated IDs

From these lists the menu system configuration data can describe the menus using the following terminology.

There are a number of **Physical Buttons** that are user interactive items like soft-keys and hard-keys. These are the points at which the user can interact.

There are a number of **Logical Buttons** that represent the Menu Events that can be sent to the applications. A Logical Button is defined with text, icons, and styles determined by the integrator and possibly altered by one or more Application Status items.

A **Menu** links logical buttons to physical buttons. A menu can be displayed based on the current Page, and possibly controlled by Application Status data.

Within the configuration file a definition for the characteristics of each logical button is established, including conditions for styling based on inputs. Menus are developed as assignments of defined logical buttons to physical buttons; menus might have conditions for display based on Application States. Pages are assigned a list of possible menus, with a priority for which menu would be displayed over others.

This system of configuration has proven its use on CMS by allowing rapid changes to the available options as requirements evolve, capability windows are re-arranged, and priorities for our demonstrations have shifted.
A Common Command Interface for Interactive UoCs

Menu System Interface

The concept of interactive UoC communication to a Menu System, if widely used, promotes UoC reuse. It brings independence from the end-system’s user interface solution. This interface has two basic concepts:

1. Applications publish the Application State Data associated with its user inputs. This allows the menu to properly render the state of these inputs per the integration style of the menu.

2. Applications can receive Menu System Events when the user selects the menu widget associated with that event.

The CMS Menu System has modeled messages for menu event and application state data. This gives a standard well defined way for applications to interact with the menu. The message models do not have to be the same as those depicted here. The TSS marshalling and transformations can handle these differences, but the concepts of sending status and receiving events should be built into UoCs that have user interactions.

Menu System Event Message

An application receives Menu System Events from the Menu UA. For CMS this message includes an ApplicationID and an EventID.

The ApplicationID can be used by the application to confirm the message, or to identify the message as belonging to the application if the menu system shares a topic with multiple applications.

The EventID identifies which of the events an application supports was triggered within the menu.

```
namespace FACE
{
    namespace DM
    {
        struct AC_MenuEvent
        {
            FACE::DM::tApplicationID ApplicationID;
            FACE::DM::tEventID EventID;
        }
    }
}
```

Figure 6- Menus System Event Message

Application State Data Message

An application publishes the Application State Data Message periodically or when states change. For CMS this message includes an ApplicationID, a StateID, and a CurrentState.

The ApplicationID identifies the UoC.

The StateID identifies which of the published states this message is addressing.

The CurrentState contains the current state.

```
namespace FACE
{
    namespace DM
    {
        struct AC_ApplicationStateData
        {
            FACE::DM::tApplicationID ApplicationID;
            FACE::DM::tStateID StateID;
            FACE::DM::tCurrentState CurrentState;
        }
    }
}
```

Figure 7 - Application State Data Message
A Common Command Interface for Interactive UoCs

Conclusion

The CMS team discovered early in its development that a configurable menu system, independent of the interactive UoCs that would use it, is critical to presenting a common look and feel to the primary user interface within the CMS. Through use of the configurable features of this system, CMS has been able to quickly respond to user interface changes made from user feedback and the addition of new capabilities.

The interface between the menu system and the UoCs allows CMS to add simple interfaces to UoCs that otherwise do not have a user interface. For example CMS currently uses a fixed non-controllable camera. In the future, CMS might use a camera that has mode and field of view settings which could be implemented in a PSSS Device Service without the need to implement a full ARINC-661 interface to that UoC.

The software supporting the menu system is flexible enough to support major changes in the menu format and style. If CMS were to switch displays the menu could be changed drastically without modification of this high-criticality source code. For instance, a tablet without bezel buttons could have the menu use pop-ups from the bottom edge.

As CMS positions itself to accept UoCs developed by other projects, CMS hopes those other projects are employing a similar concept. Wide adoption of this menu system interface pattern will greatly improve portability of UoCs within the FACE community.
References

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

- FACE Technical Standard 2.1
- DO-178C
- DoD Joint Software Systems Safety Engineering Handbook
- MIL-STD-1472G
- ARINC 661
- ARINC 661-6
A Common Command Interface for Interactive UoCs

About the Author(s)

Steven P. Price has been working in avionics and embedded software for 30 years. He’s worked on several different graphic user interfaces including cockpit systems. He’s been a leader in the design and implementation of some of these systems, along with being involved with the testing of some of these systems. Currently Mr. Price is one of the Software Engineers for CMS, and the principal developer of the CMS Menu System. He is a FACE Verification Authority Subject Matter Expert (SME).

Christopher J. Edwards has been working in the avionics industry for 20 years, mostly on cockpit systems for military aircraft. In those years he has served in leadership roles in Software, Requirements, System Design, PVI development, Qualification Testing, and Project Management. Mr. Edwards has been the primary author of the FACE Conformance Certification Guide and the Problem Report/Change Request (PR/CR) Process and a contributor to several other documents in both the Technical Working Group (TWG) and Business Working Group (BWG). Mr. Edwards currently serves as the Lead Systems Engineer for the CMS Project and as a FACE Verification Authority Subject Matter Expert (SME).
A Common Command Interface for Interactive UoCs

About The Open Group FACE™ Consortium

The Open Group Future Airborne Capability Environment (FACE™) Consortium, was formed in 2010 as a government and industry partnership to define an open avionics environment for all military airborne platform types. Today, it is an aviation-focused professional group made up of industry suppliers, customers, academia, and users. The FACE Consortium provides a vendor-neutral forum for industry and government to work together to develop and consolidate the open standards, best practices, guidance documents, and business strategy necessary for acquisition of affordable software systems that promote innovation and rapid integration of portable capabilities across global defense programs.

Further information on FACE Consortium is available at www.opengroup.org/face.

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 500 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

• Capture, understand, and address current and emerging requirements, and establish policies and share best practices
• Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
• Offer a comprehensive set of services to enhance the operational efficiency of consortia
• Operate the industry’s premier certification service

Further information on The Open Group is available at www.opengroup.org.