Extending a Flight Management Computer for Simulation and Flight Experiments

Michael M. Madden[*]
NASA, Hampton, VA, 23681

and

Paul C. Sugden [†]
Unisys Corporation, Hampton, VA, 23666

In modern transport aircraft, the flight management computer (FMC) has evolved from a flight planning aid to an important hub for pilot information and origin-to-destination optimization of flight performance. Current trends indicate increasing roles of the FMC in aviation safety, aviation security, increasing airport capacity, and improving environmental impact from aircraft. Related research conducted at the Langley Research Center (LaRC) often requires functional extension of a modern, full-featured FMC. Ideally, transport simulations would include an FMC simulation that could be tailored and extended for experiments. However, due to the complexity of a modern FMC, a large investment (millions of dollars over several years) and scarce domain knowledge are needed to create such a simulation for transport aircraft. As an intermediate alternative, the Flight Research Services Directorate (FRSD) at LaRC created a set of reusable software products to extend flight management functionality upstream of a Boeing-757 FMC, transparently simulating or sharing its operator interfaces. The paper details the design of these products and highlights their use on NASA projects.

Acronyms

1U	=	One Unit ("unit" is a standard unit of height for electronic system racks.)
ACARS	=	Aircraft Communications Addressing and Reporting System
ARIES	=	Airborne Research Integrated Experiments System
ARINC	=	Aeronautical Radio Incorporated
ATAAS	=	Advanced Terminal Area Approach Spacing
B757	=	Boeing 757
CDLS	=	CTAS-Data Link System
CDU	=	Control and Display Unit
CLB	=	Climb page menu key
CPI	=	Control Panel Interface
CRZ	=	Cruise page menu key
CTAS	=	Center-TRACON Automation System
DES	=	Descent page menu key
EFIS	=	Electronic Flight Instrument System
FANS	=	Future Air Navigation System
FMC	=	Flight Management Computer
FMS	=	Flight Management System
FRSD	=	Flight Research Services Directorate
IEI	=	Imbedded Element Identifier
IMI	=	Imbedded Message Identifier
LaRC	=	Langley Research Center

[*] Aerospace Engineer, Flight Simulation and Software Branch, Mail Stop 125B, Senior Member AIAA.
[†] Software Engineer, Federal Systems Division, 20 Research Drive.

LNAV	=	Lateral Navigation
LNG	=	Low Noise Guidance
LSK	=	Line Select Key
PC	=	Personal Computer
PIP	=	Product Improvement Package
TRACON	=	Terminal Radar Approach Control
TRS	=	Transport Research System
VNAV	=	Vertical Navigation

I. Introduction

Computers continue to drive productivity and increase customer value in many segments of the U.S. economy. The aviation industry is no different. Airlines have long used Flight Management Computers (FMCs) to minimize cost by optimizing fuel consumption and travel time. Current needs to improve flight safety and security and reduce congestion in the National Airspace System are driving research into new technology solutions that incorporate the FMC. These solutions expand the FMC into an information hub between the pilot, ground stations, and other aircraft. The research requires expanded capabilities for the FMC. FMC's are closed, proprietary systems. An FMC cannot be modified without partnering with the manufacturer; partnering is not always practical. Developing a tailorable software product that replicates the functionality of a modern, transport FMC may not be a viable solution. Domain knowledge is scarce; only a handful of FMC manufacturers exist. Even with sufficient domain knowledge, such a complex system requires millions of dollars and several years of development to replicate. The Flight Research Services Directorate (FRSD) at NASA Langley Research Center has undertaken such a development project called the NASA Research Flight Management System. The system has taken five years of development and is near completion. In the interim, however, FRSD needed to support advanced Flight Management System (FMS) research. Thus, FRSD developed a system that leverages the existing features of an FMC and runs software upstream of the FMC to provide extended features. The FMC extension software must seamlessly integrate with the FMC from the perspective of human operators (e.g. pilot). The FMC extension software must interoperate with the FMC using the FMC's existing interfaces.

The Boeing 757 FMC has three primary interfaces that communicate information: the Electronic Flight Instrument System (EFIS), the Control and Display Unit (CDU), and data link. The FMC drives the navigation display on the EFIS. The navigation display depicts the route that is programmed into the FMC along with other geographically-mapped and general information.[‡] The CDU provides a screen and keyboard. Generally, textual information is shown on the screen but modern CDUs are also capable of displaying graphics. The text information is organized into 'pages' that the pilot can access via a key press.[§] The collection of available pages in the FMC is called a page set. Pilots use the CDU to make entries into the FMC and view FMC data. Data link provides communication between the FMC and a ground station. Data link allows information transfer between the pilot and airline operations centers or air traffic controllers. FRSD has developed software components that utilize the EFIS, CDU, and data link interfaces to extend the FMC's features from the perspective of the pilot and ground stations. The software accomplishes the extension through three primary mechanisms: 1) seamless augmentation and replacement of display information from the FMC to the EFIS and CDU, 2) automated manipulation of the FMC via CDU keystroke emulation and data link insertion to support the activation of extended features, and 3) extracting data from the data link, EFIS, and CDU connections.

II. System View

Augmenting the FMC with external software begins with the system design. Figure 1 depicts a system view of the FMC within FRSD's facilities. The FMC is connected to a VME chassis. This connection includes ARINC-429 digital buses, discrete lines, and power. This VME chassis is an enclosed VME back plane with a Motorola PowerPC processor board, a SCRAMNet+[¶] module, a discrete input and output card, and two Condor ARINC 429 cards. The VME chassis is connected to the SCRAMNet+ network and mediates the transfer of data between the FMC and the SCRAMNet+ network. The *research computer* is an SGI Origin that is also connected to the SCRAMNet+ network. In the simulator, the research computer hosts the simulation program; on the aircraft, the

[‡] Examples of geographically-mapped information are airports and waypoints. Examples of general information are airspeed and heading.

[§] This key press is either a menu key or a line select key from another page.

[¶] SCRAMNet+ is a reflective-memory, fiber-optic network that utilizes a ring topology.

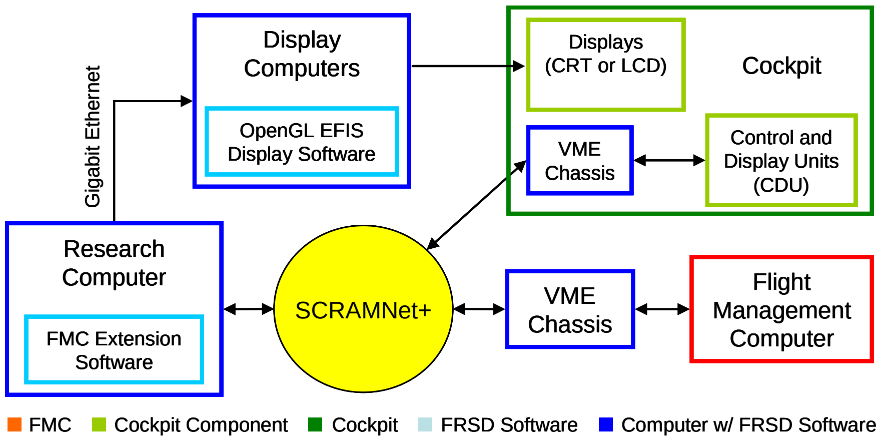

Figure 1 FMC in the System

research computer hosts the flight research program. The FMC extension software runs in a separate process or embedded in the simulation or flight research program. The *display computers* are a rack of one-unit (1U) personal computers (PC) running Linux on Intel processors. The display computers contain OpenGL-based programs that render displays in the cockpit; the EFIS is one of the programs. The display computers receive data from the research computer via gigabit Ethernet. The display computers' video outputs are distributed from the rack to the displays in the cockpit. The cockpit is also connected to the SCRAMNet+ network via a VME chassis. The VME chassis is similar to the chassis connected to the FMC but has additional I/O cards. The cockpit's chassis directs data transfer between the cockpit instruments and the SCRAMNet+ network. The CDU is one of these instruments.

FRSD designed this system with flexible communication routing that provides the foundation for the FMC extension software. Each path of communication (e.g. an ARINC-429 bus) is dynamically assigned an address in SCRAMNet+ at runtime. The research computer controls the allocation of addresses in SCRAMNet+. The allocation can be programmed by a developer or determined by a configuration file. The following is a simplified description of how the system establishes the connections; more detailed information can be found in references [1] and [2]. The research computer writes the configuration information into a table at the start of SCRAMNet+. The research computer then sets a word in SCRAMNet+ that signals the other computers and VME chasses on the network to read the table and configure themselves to read from or write to the assigned address. For example, the operator could configure the address for the FMC's CDU output such that cockpit VME reads the data and transmits it to the CDU. This is the configuration for an FMC without augmentation; the FMC communicates "directly" with the CDU. When the FMC will be augmented by FMC extension software, the operator configures the address for the FMC's CDU output to connect to the research computer and the operator configures the cockpit VME with an address for CDU input that originates from the research computer. Thus, CDU communication is routed through the research computer where the research program can augment or replace the data. The next sections discuss the reusable components that projects leverage to create FMC extension software that manipulates the re-routed data.

III. FMC Extension Components

FRSD has developed three categories of extension components that target the three main interfaces of the FMC: 1) CDU components, 2) Open-GL EFIS, and 3) data link components. FRSD created the components using object-oriented C++. The CDU components provide basic capabilities to create custom page sets, to arbitrate control of the CDU between the FMC and the FMC extension software, to capture and modify the content of CDU pages, to emulate CDU keystrokes, and to capture flight plan data that the B757 FMC sends to the CDU. (The flight plan data is not part of the display data; it is numeric and text data that the B757 CDU can use to drive the map display if the FMCs fails.) The OpenGL-based EFIS product replaces the EFIS line replaceable unit (i.e. black box). The OpenGL EFIS interface allows limited drawing of custom items on the display, and the product architecture supports tailoring for more complex customizations. The data link components allow data link messages to be transferred to or from the FMC. Data link messages can automate data insertion or data extraction from the FMC.

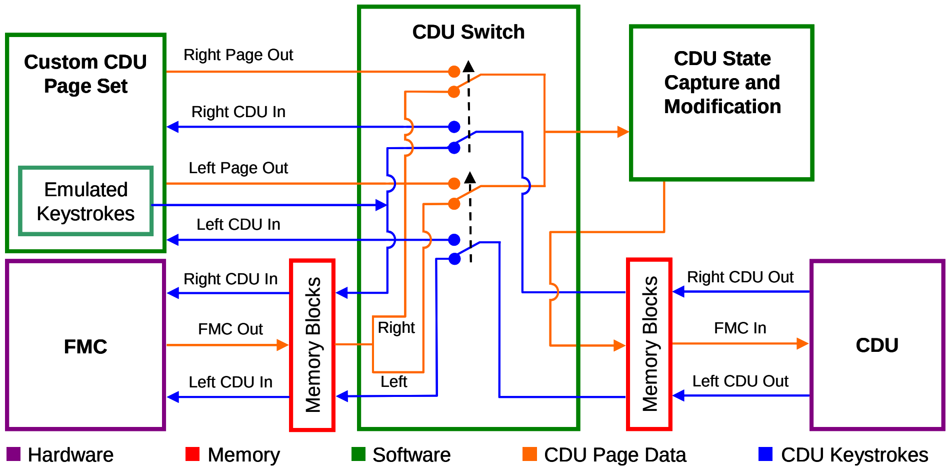

Figure 2 Conceptual Diagram of CDU Components

A. CDU components

The CDU components are a set of collaborating classes that manipulate the stream of CDU data emitted by the FMC and the keystrokes emitted by the CDU. Figure 2 depicts a simplified, conceptual depiction of the CDU components and how they interact with the FMC and CDUs. The CDU components are divided into four packages: 1) a custom CDU page set, 2) a CDU switch, 3) CDU state capture and modification, and 4) flight plan components (not shown). The custom CDU page set provides a CDU interface to the extended FMS features. The CDU switch arbitrates control of each CDU between the FMC and the custom CDU page set. The CDU state capture and modification component captures the CDU state emitted by the FMC (i.e. page, scratchpad, and annunciators) and allows a project to read or manipulate the state before it is sent to the CDU. The flight plan component decodes and stores the flight plan data the B757 FMC sends to the CDU.

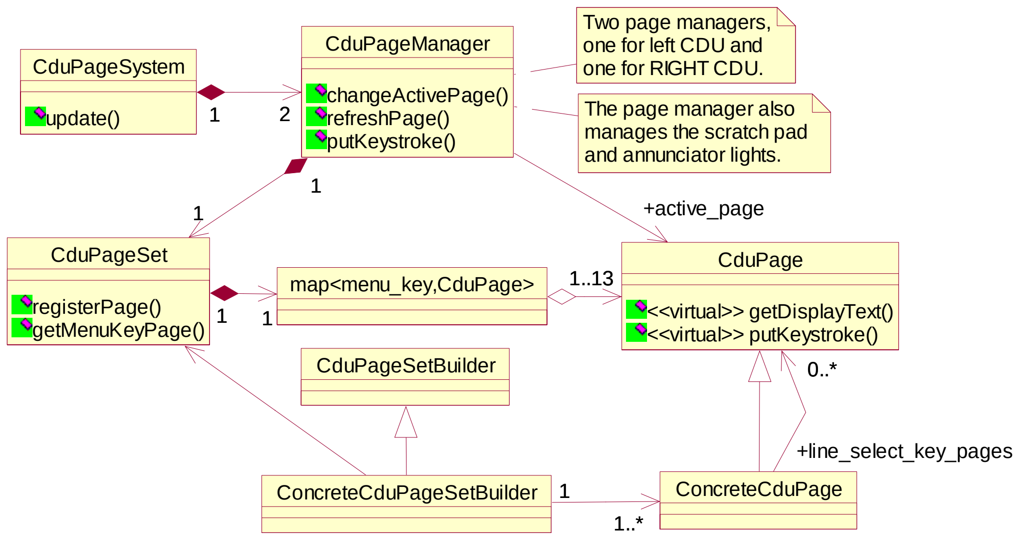

Figure 3 Top-Level Architecture of the Custom CDU Page Set

1. Custom CDU Page Set

Figure 3 depicts the top level architecture of the Custom CDU Page Set in Unified Modeling Language notation. Operationally, the top object is the CduPageSystem. The CduPageSystem connects CduPageManager objects to an I/O interface such as the CDU Switch software. By default, it connects two CduPageManagers, one for the left CDU and one for the right CDU. CduPageSystem::update() causes the CduPageManagers to process any keystroke inputs from the I/O interface and to upload page displays (including scratchpad or annunciator light updates) to the I/O interface. The CduPageManager operates the collection of pages embodied in the CduPageSet object. The

CduPageManager encapsulates the dynamic state of CDU, i.e. which page is active, what annunciator lights are active, and what are the contents of the scratchpad. The CduPageSet represents the static properties of the page set, i.e. what pages exist in the set and which page becomes active when a menu key is pressed.

Both the CduPageManager and CduPageSet rely on the object-oriented feature of abstract interfaces to operate a set of heterogeneous pages without knowing the purpose of each page. All CDU pages derive from a common ancestor, CduPage. CduPage is an abstract interface that provides the basic methods for manipulating a page. For example, getDisplayText() renders the page, and putKeystroke() responds to operator inputs. A concrete CduPage (e.g., the INDEX page) uses polymorphism to define the abstract methods with the page's specific implementation. Thus, when the INDEX page is the object assigned to the active_page in the CduPageManger, the CduPageManager will receive the content of the INDEX page when it calls active_page→getDisplayText(). A concrete subclass of CduPageSetBuilder determines the actual collection of pages, with which the CduPageSet and CduPageManager interact. In other words, the CduPageSetBuilder subclass is the only class that knows what concrete CduPage subclasses will be in the page set. It constructs each concrete CduPage and registers the object to the CduPageSet as a CduPage pointer; it also configures the mapping of menu keys to CduPage objects.

The concrete CduPages also take advantage of the abstract CduPage interface to anonymously connect with other concrete CduPages for performing tasks such as a page transition in response to the operator pressing a line select key (LSK). For example, the INDEX page (e.g. class name "IndexPage") may contain a link to the TAKEOFF page from LSK 4L.[#] The IndexPage stores this link as a pointer to a CduPage, not a pointer to a concrete TakeoffPage. This design provides the freedom to link the IndexPage to any number of TAKEOFF page implementations without changing the IndexPage code. The IndexPage can therefore be reused in multiple page sets, each with a different TAKEOFF page implementation. Developers can extend an existing page set or modify the "navigation map"[**] of the page set by adding or modifying a few lines of code in the CduPageSetBuilder subclass. This property of the design allows research projects to rapidly prototype an experimental page set from an existing page set or from individual CduPage subclasses.

2. The CDU Switch

The CDU switch performs independent switching of the left and right CDUs. For example, the custom CDU page can control the left CDU while the FMC controls the right CDU. Figure 2 shows three external connections for CDU data, two keystroke connections and one page connection. In the Boeing 757 system, the FMC has independent inputs for incoming keystrokes but combines the page output for all CDUs into a single output. To support a "direct" connection between FMC and CDU in FRSD's facilities, the VME chassis in the cockpit (see Figure 1) uses a single memory block for incoming CDU page data but separate memory blocks for keystroke data. The CDU switch software must, therefore, split the FMC output into left (pilot) and right (copilot) in order to perform the independent switching. The CDU switch must also recombine the two data streams following source selection. The CDU switch uses menu keys to determine which page source has control over a CDU. The developer can configure the CDU switch to map one or more menu keys to the custom CDU page set. When the operator presses one of the mapped menu keys, control over the operator's CDU is switched to the custom CDU page set. When the operator presses a menu key that is not mapped, control is returned to the FMC.

The CDU switch also provides a connection that allows the custom CDU page set to insert keystrokes into one of the FMC's CDU inputs. The CDU switch uses an FMC CDU input that corresponds to a CDU which is not currently controlled by the FMC. Being a user interface, the Custom CDU Page Set sends emulated keystrokes only in response to operator actions; the Custom CDU Page receives operator input only if it controls one of the CDUs. Therefore, emulated keystrokes can only be emitted if the Custom CDU Page controls at least one CDU.

[#] LSK 4L is the fourth line select key from the top and on the left.
[**] The "navigation map" is the path of page connections via menu keys and line select keys.

3. *CDU State Capture and Modification*

The CDU state capture and modification component captures the current CDU state (e.g. page, scratchpad, annunciators, and last key press) and decodes it into text, boolean, and integer values that the FMC extension software can read and/or manipulate. The state capture software operates on both the FMC-generated and custom CDU pages. Thus, the component can generate a log of all CDU page views, annunciator lights, and pilot keystrokes. By reading the CDU state, the FMC extension software can extract FMC information that is only available from CDU pages. Furthermore, the FMC extension software can alter any portion of the FMC-generated CDU state to incorporate extended FMS features into the FMC's CDU output.

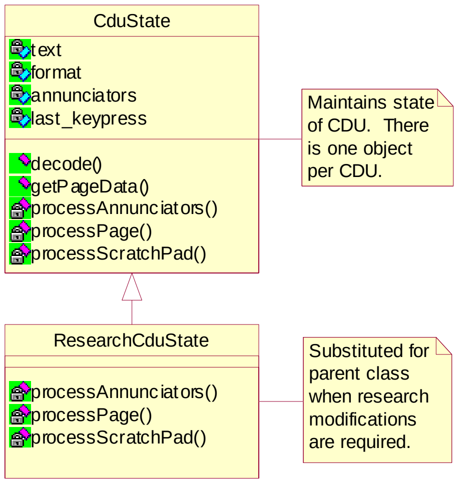

Figure 4 CDU State Design

Unlike the custom page software, modifications to the CDU state do not require switching of the CDU I/O. The CDU State component is unaware of the source of state data or the destination of CDU key presses. Figure 4 depicts the primary classes of this component. The CduState class decodes and stores the CDU state; there is one CduState object per CDU. Each object contains data structures representing the page text and formatting, on/off status annunciator lights, and the most recent key press made on the CDU. Private, virtual methods execute state modification; these methods are named processAnnunciators(), processPage(), and processScratchPad(). The CduState class invokes the appropriate method whenever that portion of the CDU state is updated. The default methods in the CduState class are empty. Research-specific classes derived from CduState can provide implementations for these methods that modify specific portions of the state data before it is transmitted to the CDU.

4. *Flight Plan Components*

The flight plan components decode flight plan information that the Boeing 757 FMC sends to the CDU. The Boeing 757 FMC interlaces this data with the CDU page data. In the event that the FMCs fail, the B757 CDUs can use the last uploaded flight plan to generate their own pages and drive the navigation displays. FRSD's system separates the flight plan data into a dedicated SCRAMNet+ memory block. The FmcFlightPlan object reads this data and decodes it into a vector of FmcWaypoint objects. The flight plan data from the FMC contains only the latitude, longitude, and identifier for each waypoint. The CDU Page Capture and Modification component (see III.A.3) provides the option of populating the speed and altitude attributes of the FmcWaypoint class by scanning the FMC's LEGS page. FMC extension software can use the flight plan data as a source of information for its algorithms or to repackage the information for new displays or new external communication (e.g. Distributed Air/Ground Traffic Management [DAG-TM]).

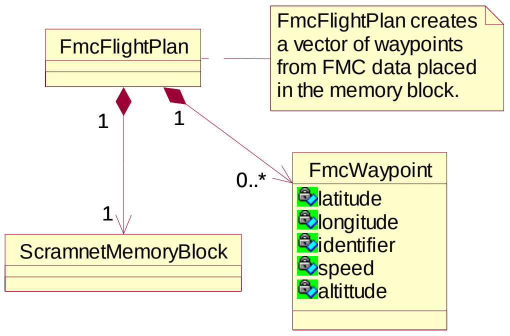

Figure 5 Class Diagram of the Flight Plan Components

B. OpenGL EFIS Software

FRSD's system replaces the black-box EFIS with display software that draws the EFIS display using OpenGL. Figure 6 depicts a conceptual view of the OpenGL EFIS software in the system. The display software uses a memory-based interface that is populated with symbol data for the display. The symbol data is composed of intrinsic types; it is not a stream of ARINC words such as the FMC emits. The ARINC 702 Translator on the research computer retrieves the FMC's EFIS output from SCRAMNet+ and decodes it into the display software's interface. In FRSD's prior-generation system architecture, the display software also existed on the research computer; the translator and display software communicated directly via shared memory. In FRSD's current system architecture, the display software has been offloaded to a dedicated display computer (1U PC). The research computer now communicates the symbol data in the memory block over gigabit Ethernet using sockets, and the display computer takes the socket data and places it in a memory block. FRSD retained the memory interfaces for the flexibility of running the display software in configurations that used shared memory or SCRAMNet+ for communication of symbol data.

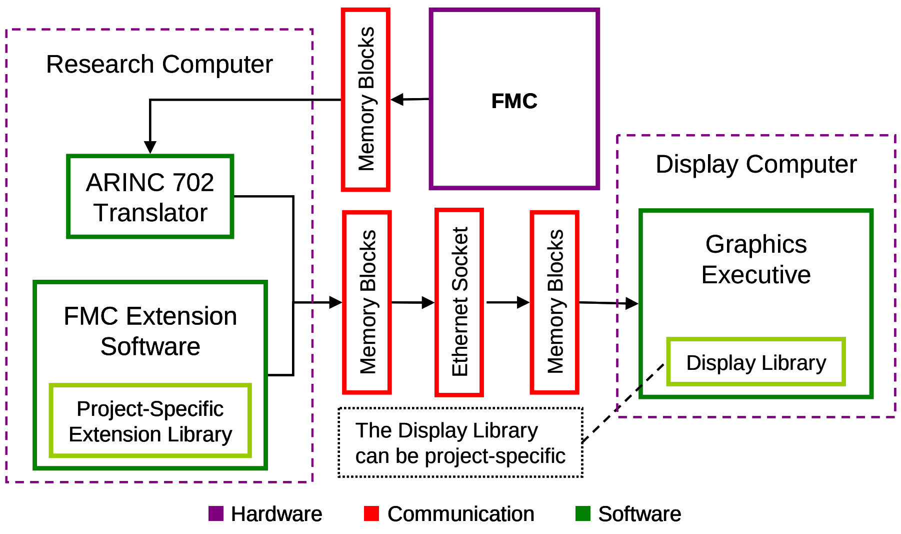

Figure 6 Conceptual View of OpenGL EFIS Software

The FMC Extension Software can access the memory block for symbol data in order to modify or augment the display. Unlike the CDU, no generic feature to switch between the FMC and FMC extension software is needed. The display communication is unidirectional and the FMC Extension Software usually augments existing displays rather than replace them. Even if replacement is needed, the FMC data can simply be ignored.

The project can make limited augmentation of the standard navigation display by deriving from the ARINC 702 translator and inserting its own data. This technique works best when the additional data can be displayed using existing symbols in the standard navigation display. Creating new symbols requires tailoring the display software. The architecture of the display software offers the flexibility to easily install tailored displays. The display software consists of a graphics executive that can dynamically load a display library which adheres to a standard export interface. When the simulation starts, it sends data (via sockets) to the display computers; upon reception of this data, the display computer initiates the launch of the graphics process. The simulation then provides the pathname of the library that the graphics executive will dynamically load and execute. When the simulation ends, the simulation sends a data signal (via sockets) that causes the graphics process to terminate.

The display libraries are written in object-oriented C++. FRSD has developed a reusable architecture and a repository of reusable classes for the display libraries. Tailored displays generally derive from the reusable architecture and build their extensions from the reusable classes. However, it is possible to build a new graphics library from scratch as long as the library adheres to the export interface that the graphics executive expects.

C. Data Link Components

The FMC extension software can use data link messages to automate the insertion of information into the FMC. This purpose sometimes requires the assistance of emulated CDU keystrokes to automate acceptance of the data by the FMC. The FMC extension software can also use data link messages to retrieve information from the FMC. The data link components (see Figure 7) provide basic services for encoding or decoding data link messages and transferring messages between the FMC and FMC extension software. The data link components consist of two packages, the AcarsManagementUnit[††] and the ImbeddedMessageIdentifier. The AcarsManagementUnit manages the communication of data link strings with the FMC. The package translates between plain text strings used by the FMC extension software and the protocol used by the FMC. The package receives and exports its data through instantiations of the DataHandler template. DataHandler is an abstract interface for communication that is designed for anonymous chaining. DataHandler-based interfaces can be mated to a variety of DataHandler implementations without knowledge of how the implementation will use the data (output chains) or how the implementation obtained the data (input chains). For example, the text input of the AcarsManagementUnit code can be attached to a package producing data link weather updates or to a package that produces canned data link messages for testing the connection to the FMC's data link port. The DataHandler interface allows any project to attach their FMC extension software to the AcarsManagementUnit. It also allows the project to adapt the AcarsManagementUnit to any representa-

[††] ACARS is an acronym for Aircraft Communications Addressing and Reporting System. It is the data link standard used by the Boeing 757 FMC.

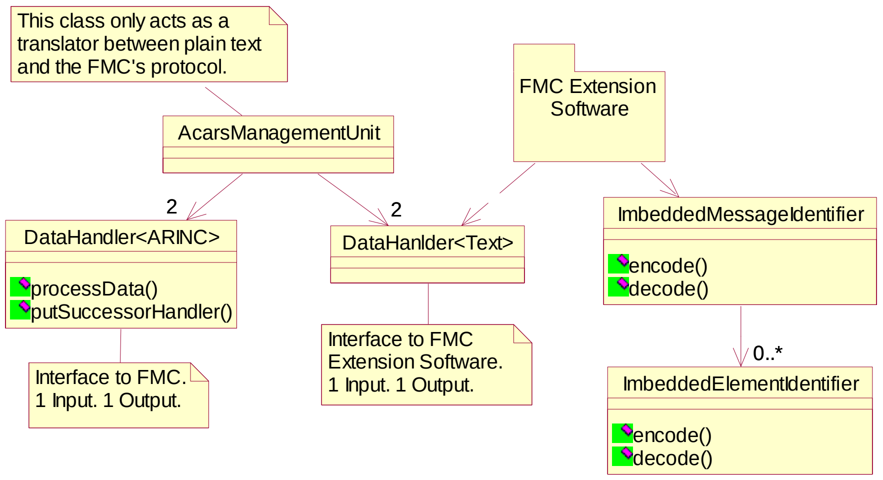
This class only acts as a translator between plain text and the FMC's protocol.
FMC Extension Software
AcarsManagementUnit
2
2
DataHandler<ARINC>
processData()
putSuccessorHandler()
DataHanlder<Text>
ImbeddedMessageIdentifier
encode()
decode()
Interface to FMC. 1 Input. 1 Output.
Interface to FMC Extension Software. 1 Input. 1 Output.
0..*
ImbeddedElementIdentifier
encode()
decode()

Figure 7 High-Level Architecture of Data Link Components

tion of FMC communication. In the FRSD facility, the DataHandler subclasses for ARINC communication implement SCRAMNet+ communication to the FMC's VME chassis. [Note: The CDU components also use DataHandler objects; however, these objects appear at a lower level of detail than shown in this paper.]

A data link message is a plain text encoding of numeric, enumerated, boolean, and text data. The ImbeddedMessageIdentifier package performs the encoding and decoding of data link strings. A data link string is composed of an imbedded message identifier and zero or more imbedded element identifiers. The design reflects this decomposition. The ImbeddedMessageIdentifier (IMI) and ImbeddedElementIdentifier (IEI) are abstract classes that encapsulate the common attributes and behaviors of both identifiers. Classes derived from ImbeddedMessageIdentifier implement a specific IMI. An ImbeddedMessageIdentifier subclass defines the text marker for the IMI, encodes/decodes the IMI fields, constructs the IEI objects that can appear in the message, and knows which IEIs are mandatory and which are optional. Classes derived from ImbeddedElementIdentifier implement a specific IEI. An ImbeddedElementIdentifier defines the text marker for the IEI, encodes/decodes the IEI fields, and knows which fields are mandatory and which are optional. The FMC extension software uses the ImbeddedMessageIdentifier package to encode/decode data link messages exchanged with the FMC. The package can also be used to create custom data link messages for communication with research products that create data link extensions to the FMC.

IV. Case Studies

This section contains examples of how the FMC extension components have been used on projects. The projects highlighted are Boeing 757 Center-TRACON Automation System (CTAS), Advanced Terminal Area Approach Spacing (ATAAS), and Low Noise Guidance (LNG).

A. Boeing 757 Center-TRACON Automation System (CTAS)

This simulation study had two objectives: 1) explore options for interfacing CTAS[‡‡] with an FMC to exchange trajectory data and 2) evaluate candidate procedures for pilot and controller interactions with the FMS and CTAS. B757 CTAS was the first project to integrate a Honeywell B757 FMC-PIP (Product Improvement Package) with LaRC's Boeing 757 Simulation. Figure 8 depicts the software and hardware interfaces in the B757 CTAS product. The development team created a software system called the CTAS-Data Link System (CDLS) that contained the extended FMC functionality that interfaced with the CTAS product. The CTAS software communicates with the Boeing 757 using plain text data link messages and structures of aircraft state information. To satisfy the first objective of the experiment, CDLS required access to the FMC's trajectory data in order to pass that data to CTAS via a plain text data link message. The research team obtained a proprietary device from Honeywell called a Control Panel Interface (CPI) that exposed the flight plan memory of the FMC. The memory contents were transmitted to CDLS which encoded the data into a data link message and transmitted the message to CTAS. In future experiments, the CPI interface would be replaced with a combination of the Flight Plan components and capturing of in-

[‡‡] CTAS is a software product developed at Ames Research Center. CTAS is a collection of aids that enable air traffic controllers to manage the increasingly complex air traffic at large airports.

formation from the FMC's LEGS page (using CDU page capture and modification). This software-only solution results in a less complex system and would require less scrutiny for deployment on LaRC's research aircraft.

The research team decided to implement the pilot interface with CTAS using modified FANS (Future Air Navigation) CDU pages that are an option on the Boeing 747. The FANS CDU pages were implemented using the CDU Custom Page System components (see III.A.1). The CDU Switch component arbitrated control of the CDU between the custom pages and the FMC pages. The CDU's cruise (CRZ) menu key was changed to an ATC key and the CDU Switch was configured to map this key to the FANS pages. The CDU switch was also tailored to cycle through the FMC's Cruise, Climb, and Descent pages when the CDU's climb (CLB) menu key was pressed. (On the face plate, the descent key was changed to a blank and the climb key was changed to VNAV.) The FANS pages connected to the CTAS-Data Link System. The pages sent commands to the system in response to pilot inputs and retrieved data from the system for display.

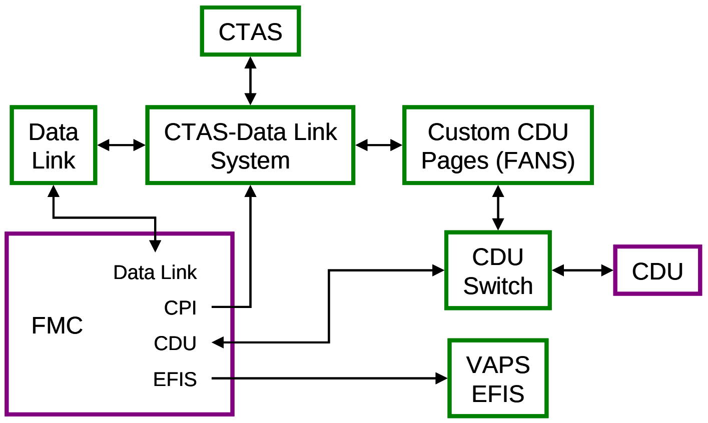

Figure 8 B757 CTAS System

CTAS sent updated weather and route modification data link messages to the Boeing 757. The pilot reviewed and accepted or rejected these uplinks via the FANS pages. To provide the illusion that the FANS pages were integrated with the FMC, the CDLS had to automatically insert information from accepted data link messages into the FMC. The CDLS packaged the data into ACARS data link messages and transmitted them to the FMC. The CDLS would follow the transmission with a series of emulated CDU keystrokes that commanded the FMC to accept the data link message. This activity exposed one of the weaknesses of extending the FMC with external software. FMC processing of the ACARS message was not deterministic and neither were the FMC's response to keystrokes. A delay had to be added between the sending of the ACARS message and the first keystroke. Some keystrokes relied on page changes commanded by a prior keystroke. A delay had to be placed between keystrokes to assure that the FMC had sufficient time to respond to the prior keystroke and update the page. The delays necessary to obtain greater than 95% success for auto-loading the data added up to several seconds in some cases. Since the emulated keystrokes were sent via the FMC input for the CDU that initiated the action, any attempt by the operator to immediately switch to an FMC controlled page would be met with a delay while the emulated keystrokes were processed. Other events could interfere with the emulated keystroke sequence, particularly if the sequence required data insertion via the scratch pad. The required delays between keystrokes widened the window for such interference. A great deal of time was spent optimizing the delays so that these issues became a rare annoyance.

The B757 CTAS project also used the standard Navigation Display available from a predecessor to the OpenGL EFIS software that used VAPS™ from Enginuity Technologies as the rendering engine; thus, the EFIS software only required input data from the FMC.

B. Advanced Terminal Area Approach Spacing (ATAAS):

Advanced Terminal Area Approach Spacing (ATAAS) explored in-trail, self-spacing of aircraft in the airport terminal area using the time-history concept. In time-history self-spacing, each aircraft attempts to fly the speed profile of the aircraft that it is following. To achieve this concept, ATAAS introduced new autothrottle and vertical navigation (VNAV) modes. Before either of these modes could be engaged, several inputs from the pilot were required, including the target-to-follow, desired spacing interval, planned final approach speed for ownship and target, and wind speed and direction. Two new CDU pages that allowed pilot input of these parameters were created using the Custom CDU Page System. One of these pages also allowed the crew to monitor current status of the spacing and speed of the lead aircraft. The CDU Switch was employed to arbitrate control of the CDU between the two custom pages and the FMC. As with CTAS, the cruise (CRZ) menu key was mapped to the custom page source and all other function keycodes caused the FMS to resume its role as the active source. However, rotating among descent, climb, and cruise pages when pressing the climb key was not implemented for this experiment; all three pages are accessible from each other normally via a line select key.

The ATAAS software intercepted the VNAV autopilot output from the FMC and modified it to drive the autothrottle in a manner that minimized in-trail spacing error upon landing. For pilot awareness, two new display concepts were required for the Navigation display: the lateral path of the target-to-follow and a separation bar indicating horizontal distance from the location that would yield the desired temporal spacing interval. The development team created a tailored display by deriving from the classes of the standard navigation display.

Portability of this software was verified when ATAAS moved from simulation study to flight demonstration. This required rehosting the custom CDU page and CDU switching software to the Transport Research System (TRS) onboard NASA's Boeing 757 Airborne Research Integrated Experiments System (ARIES). This proved possible with only minor enhancements to the TRS-FMS interface.

C. Low Noise Guidance (LNG)

The Low Noise Guidance (LNG) simulation aims to minimize noise in populated areas during aircraft approach. The Low Noise Guidance (LNG) study introduced custom Lateral Navigation (LNAV) and Vertical Navigation (VNAV) autopilot modes. LNG required the new modes to operate transparently in conjunction with the existing FMC. The new control laws required the flight plan as one of its inputs. LNG used the flight plan components (see III.A.4) to obtain a copy of the active flight plan. In addition to the active flight plan, the LNG algorithms required any modified active route. While modified active routes are computed and maintained by the FMC, the FMC does not provide them as an output. LNG also required constraints associated with flight plan waypoints; the FMC also does not output the constraints. Using the CDU capture and modification software, all "LEGS" pages visited by either pilot or first officer were scanned for mod active routes and constraint information. This information was collected and stored in the LNG research software.

Since ATAAS was incorporated as part of the LNG experiment, the same custom page generating and switching software used for the ATAAS project was also used for LNG. In addition to custom pages, this project required minor modifications to several of the FMC-generated CDU pages. Since VNAV target speeds and altitudes were modified by the LNG software, the CDU page capture and modification software was used to overwrite select information on the LEGS and PROGRESS pages with LNG’s data. Furthermore, the CDU components were tailored to block selected annunciations from the FMC and automatically clear select scratchpad messages. These steps were required to maintain the illusion of an FMS incorporating new autopilot modes.

A standard FMS VNAV path includes basic altitude profile events such as "Top of Descent", and "End of Descent". These events are generated by the FMS as part of the EFIS data stream. LNG added additional altitude profile events, such as flap and gear extensions, for the purpose of controlling noise levels on descent. The standard navigation display library already has the capability to display altitude profile points. Thus, the new points were implemented by capturing the FMC’s EFIS stream and augmenting it with the new profile points. The project developed a derivation of the ARINC 702 Translator (see III.B) that filtered FMC events from the EFIS symbols and augmented the display with LNG event data. This data was then sent to the standard navigation display.

V. Conclusion

FRSD has developed a set of reusable software components that reduce the effort to create extended FMS functionality around an FMC. The software has successfully been deployed on several research projects (including B757 CTAS, ATAAS, and LNG) to provide experimental FMS functionality for increasing aviation safety, aviation security, airport capacity, and environmental impact of aircraft. These components interact with the three main information interfaces of the FMC: CDU, EFIS, and data link. By manipulating the information interfaces, the software components provide the illusion of an integrated FMS system to operators. However, the FMC’s interfaces also limit the degree to which external software can extend the FMC’s function. The software cannot control how quickly the FMC responds to inputs. The software cannot fundamentally alter the FMC’s flight planning, trajectory generation, or guidance without creating full-featured replacements of these functions outside of the FMC. Over time, such replacements border on full development of a software flight management system. Thus, organizations with requirements for long-term experimentation in these areas should pursue a feature-complete software flight management system. FRSD has used the FMC extension components as a stepping stone and interim solution to a software flight management system. FRSD has designed many of the FMC extension components for reuse in the software flight management system that FRSD is developing.

Acknowledgments

The authors want to acknowledge the following individuals for their contributions to the software described in this paper: James Barnes, Arlene Guenther, Patricia Glaab, Brian Hutchinson, Jerry Karwac, Patrick “Sean” Kenney, Jason Neuhaus, Myron Sothcott, and Jerry VanPelt.

References

[1] Chung, V. I., and Hutchinson, B. K., “A Unique Software System for Simulation-to-Flight Research,” AIAA-2001-4057, *AIAA Modeling and Simulation Technologies Conference*, Montreal, August 2001.

[2] Geyer, D. W., et. al., “Managing Shared Memory Spaces in an Object-Oriented Real-Time Simulation,” AIAA-98-4532, *AIAA Modeling and Simulation Technologies Conference*, Boston, August 1999.