Deploying a Route Optimization EFB Application for Commercial Airline Operational Trials

David A. Roscoe[1], Robert A. Vivona[2], Sharon E. Woods[3] and David A Karr[4]
Engility Corporation
Billerica, MA, U.S.A.
Email: [1]david.roscoe@engilitycorp.com, [2]robert.vivona@engilitycorp.com, [3]sharon.woods@engilitycorp.com, [4]david.karr@engilitycorp.com

David J. Wing
NASA Langley Research Center
Hampton, VA, U.S.A.
Email: david.wing@nasa.gov

***Abstract*—The Traffic Aware Planner (TAP), developed for NASA Langley Research Center to support the Traffic Aware Strategic Aircrew Requests (TASAR) project, is a flight-efficiency software application developed for an Electronic Flight Bag (EFB). Tested in two flight trials and planned for operational testing by two commercial airlines, TAP is a real-time trajectory optimization application that leverages connectivity with onboard avionics and broadband Internet sources to compute and recommend route modifications to flight crews to improve fuel and time performance. The application utilizes a wide range of data, including Automatic Dependent Surveillance Broadcast (ADS-B) traffic, Flight Management System (FMS) guidance and intent, on-board sensors, published winds and weather, and Special Use Airspace (SUA) schedules. This paper discusses the challenges of developing and deploying TAP to various EFB platforms, our solutions to some of these challenges, and lessons learned, to assist commercial software developers and hardware manufacturers in their efforts to implement and extend TAP functionality in their environments.**

EFB applications (such as TAP) typically access avionics data via an ARINC 834 Simple Text Avionics Protocol (STAP) server hosted by an Aircraft Interface Device (AID) or other installed hardware. While the protocol is standardized, the data sources, content, and transmission rates can vary from aircraft to aircraft. Additionally, the method of communicating with the AID may vary depending on EFB hardware and/or the availability of onboard networking services, such as Ethernet, WIFI, Bluetooth, or other mechanisms. EFBs with portable and installed components can be implemented using a variety of operating systems, and cockpits are increasingly incorporating tablet-based technologies, further expanding the number of platforms the application may need to support. Supporting multiple EFB platforms, AIDs, avionics datasets, and user interfaces presents a challenge for software developers and the management of their code baselines. Maintaining multiple baselines to support all deployment targets can be extremely cumbersome and expensive. Certification also needs to be considered when developing the application. Regardless of whether the software is itself destined to be certified, data requirements in support of the application and user interface elements may introduce certification requirements for EFB manufacturers and the airlines. The example of TAP, the challenges faced, solutions implemented, and lessons learned will give EFB application and hardware developers insight into future potential requirements in deploying TAP or similar flight-deck EFB applications.

Keywords— TASAR, TAP, EFB, ADS-B, ARINC 702A-1, STAP, certification, broadband, Internet, avionics, AOP

I. INTRODUCTION

The Traffic Aware Planner (TAP) was developed to demonstrate NASA's Traffic-Aware Strategic Aircrew Requests (TASAR) concept [1-3]. TASAR leverages onboard decision-support automation and connectivity to data sources onboard and external to the aircraft to produce flight-optimizing route change advisories to the flight crew that, when requested of Air Traffic Control (ATC), are more likely to be approved. TAP is the product of more than a decade of research and development with the NASA Autonomous Operations Planner (AOP) [4], an advanced cockpit automation system for airborne trajectory management that has been adapted for the TASAR project into a near-term Electronic Flight Bag (EFB) application that can provide real operational benefits for today's data-enabled aircraft. It has been deployed aboard the Avanti Piaggio P180 flight-test aircraft, owned and operated by Advanced Aerospace Solutions (AdvAero), for two NASA flight trials: Flight Trial 1 (FT-1) [5] in November 2013, and Flight Trial 2 (FT-2) in June 2015. For FT-1, TAP was installed on a UTC (United Technologies Corporation) Aerospace Systems (UTAS) G500 SmartDisplay® EFB, an Intel-based platform configured to use Windows XP. For FT-2, the processing components of TAP were installed on two Intel-based computers running Windows, while the user interface was installed on a combination of Apple iPad® Air and iPad Mini EFB devices running the iPhone Operating System (iOS).

TAP is currently being prepared for NASA operational trials with two partner airlines, Alaska Airlines and Virgin America. Alaska Airlines will host TAP on the UTAS Aircraft Interface Device (AID), which is based on the CentOS Linux kernel, the UTAS Tablet Interface Module^{TM} (TIM), which is based on the CentOS kernel on an Advanced RISC (Reduced Instruction Set Computing) Machine (ARM) platform, the Gogo Airborne Central Processing Unit (ACPU-2) and two iOS-based tablets. Virgin America will be using the Astronautics Corporation of America NEXIS™ Flight-Intelligence System EFB running a certified Linux kernel. These platform requirements, discussed in this paper, demonstrate how future EFB applications will need to deal with many variables in the environment in which it is deployed, and how considerations will have to be made with respect to code portability, computing power, installed cockpit hardware, communications, and deployment logistics.

II. The Traffic-Aware Planner

The Traffic-Aware Planner (TAP) software suite [3] consists of the following components:

A. *TAP Engine*

TAP Engine is the primary computational component of the TAP software suite. It is a distillation of the AOP research prototype cockpit automation system, developed for NASA Langley Research Center to research cockpit-based self-separation and autonomous flight operations. The role of TAP Engine is to process avionics and broadband data and to produce route optimizations that meet user-defined objectives.

B. *TAP Display*

TAP Display is the graphical Human-Machine Interface (HMI), allowing the crew to configure and control the operation of the TAP Engine. TAP Display also provides visual representation of the surrounding airspace and proposed optimization solutions.

C. *TAP Display Adapter*

TAP Display Adapter is a lightweight communications proxy for the TAP Engine and TAP Display, managing inter-process communication and maintaining state data for TAP Display.

D. *External Data Server*

The External Data Server (EDS) provides data to the TAP Engine from broadband sources for convective weather, wind data, and Special Use Airspace (SUA) schedules.

E. *TAP Utility*

TAP Utility is a standalone application for validating software installation and providing a user interface to shut down and relaunch the system, should it become unresponsive.

F. *TAP Service*

TAP Service is a lightweight daemon process launched when the aircraft is powered up. Its role is to launch the TAP system upon command from the TAP Display and to process data post-flight before the aircraft is powered off.

Fig. 1 shows the core TAP architecture and supported data sources. TAP Utility and TAP Service are not shown, as they are not intrinsic core components of the system. Instead, they are necessary to facilitate deployment of the core components.

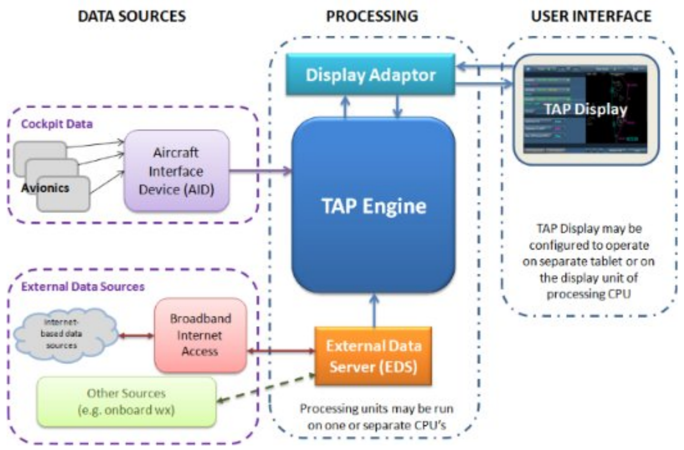
DATA SOURCES
PROCESSING
USER INTERFACE
Cockpit Data
Avionics
Aircraft Interface Device (AID)
External Data Sources
Internet-based data sources
Broadband Internet Access
Other Sources (e.g. onboard wx)
Display Adaptor
TAP Engine
External Data Server (EDS)
Processing units may be run on one or separate CPU's
TAP Display
TAP Display may be configured to operate on separate tablet or on the display unit of processing CPU

Fig. 1. TAP Core Architecture

III. Software Portability

TAP is written almost exclusively in C++. This affords a great deal of platform portability, but not without planning and maintenance. While Windows platforms are relatively stable across different versions of the operating system, the same cannot be said for the Linux or Unix distributions employed by cockpit hardware. Each distribution can introduce subtle differences, even for the same kernel version. When supporting multiple Linux and/or Unix platforms, a system should be incorporated into the building and testing process to ensure that changes which work for one distribution do not inadvertently break support for others. The TAP development team accomplished this with virtual machines configured for the versions of Linux (or a reasonable facsimile) that must be supported. Virtual machines reduce the hardware cost of building and testing for each supported platform. Testing is also being performed on the actual deployment hardware, supplied by the vendor for this purpose. Another useful tool employed for TAP development is Cmake$^{©}$. CMake is a cross-platform build-system generator. Using platform-independent scripts, CMake can generate build-system project files for any platform for which CMake has a generator. Since build-system project files are created on demand, there is no need to create, maintain, or store static project files in your version control system. If changes are needed for your build system, modifying the appropriate CMake script is all that is required to replicate those changes on all supported platforms.

Low-level platform features such as Inter-Process Communication (IPC), file input/output (I/O), memory management, and threading support, can vary from platform to platform, further complicating software design, implementation, and deployment. TAP utilizes two cross-platform frameworks: The Adaptive Communication Environment^{TM} (ACE) and Qt^{TM} to mitigate this. These frameworks provide an abstraction layer between the functional code and low-level system features, allowing relatively easy porting without affecting the functional code base. Qt has the added benefits of supporting high-performance, cross-platform graphics, resource management, and mobile device architectures. Both have licensing options ranging from open source to commercial use. The use of third-party libraries and frameworks can also complicate software design for EFB applications, as their availability on the target platform may at a reduced feature level or even non-existent. It may not always be possible to install third-party dependencies due to limitations of the target platform, licensing, and certification considerations.

IV. Tablet-Based EFBs

Tablet platforms present unique challenges to EFB software developers. Alaska Airlines uses iOS-based tablets for their EFB hardware. Because TAP is not monolithic, the limitations of current tablet-based platforms preclude hosting the entirety of the TAP suite on a tablet EFB device. Further, the absence of robust multitasking support limits the tablet EFB to running only one component of the TAP suite. The obvious choice is to port the TAP Display to the tablet, since it is the primary human-machine interface of the TAP software suite. The remainder of TAP, with the exception of the TAP Utility app, must be hosted on other computing platforms on board the aircraft.

iPhone OS (iOS) is based on the Berkeley Software Distribution kernel used by Apple's OS X operating system. The hardware is an ARM architecture. As mentioned previously, the current TAP code base is written in C++, whereas native iOS applications are typically written in Objective C. Creating a parallel code base for the iOS version of TAP Display was not desirable. TAP uses a large portion of legacy AOP code; producing an entirely new code base would have been prohibitively time-consuming and costly. The existing code base will continue to be updated and improved for all supported platforms, creating a dual maintenance situation that would be extremely cumbersome to manage. To facilitate porting the TAP Display to the iOS platform using the existing code base, the HMI portion of the TAP Display, originally written to utilize freeglut, an open source implementation of the OpenGL Utility Toolkit (GLUT), was rewritten for the Qt framework. The Qt Meta Language (QML), a derivative of JavaScript Object Notation (JSON), allows for abstraction of HMI elements, resulting in code which can be translated to native hardware rendering on any supported platform. The Qt Resource System provides a platform-independent method of storing binary data in the resulting executable, which is useful for icons, bitmaps and other application resources.

An iOS application cannot be compiled natively on the target device. Instead, it must be cross-compiled for the device on a Mac computer using Apple's Xcode development platform. Xcode's build system employs a project structure that is quite unlike those of other platforms, specifically in support of digital signing, cross-compilation tool chains for iOS target hardware, application resources, and deployment provisioning. The Qt build system, like CMake, can act as a build system generator. It has increasingly robust support for iOS and Android^{TM} platforms. For TAP, a new set of build scripts had to be created for compatibility with the Qt build system. These build scripts can be bootstrapped from the command line, or via Qt's Integrated Development Environment (IDE), Qt Creator^{TM}. The Qt build system was used to generate Xcode project files for TAP Display, utilizing the existing C++ code and avoiding the need for creating an Objective C code base. Features such as integration with the iOS Settings app is currently lacking in Qt, so some Objective C code needed to be written to handle TAP Display app settings data. The Qt build system seamlessly integrates C++/Objective C files, and it also allows for including external frameworks. It is important to note that if an application is to be distributed through Apple's App Store, the use of third-party frameworks can be severely restricted and should be thoroughly investigated prior to adopting one. This was not a problem for TAP, since primary deployment is through NASA Marshall Space Flight Center's enterprise app store, and there are no plans for Apple App Store distribution in the foreseeable future.

In addition to software porting considerations, tablet support presents other challenges that must also be addressed, such as context switching, hibernation/sleep modes, gestures, screen orientation, and electrical power. When a tablet application is context-switched from foreground operation to the background, the platform provides limited options for continued functioning of the background application. Processing power and memory is reallocated to the new foreground application. If needed, the operating system of the tablet may also unload the background application altogether to free memory. Similarly, to maximize battery life, most tablets are configured to enter a sleep or hibernate mode after an elapsed period of inactivity. The user can also cause the device to enter this state manually at any time. TAP is a real-time system that operates for the duration of a flight. If the TAP Display app is unable to communicate with the rest of the system due to context switching, hibernation, or other disruption, the rest of the system continues to operate. A mechanism was needed to allow the TAP Display to synchronize its internal state with the rest of the system upon launch or restoration of its foreground context. This is one of the reasons for developing the TAP Display Adapter. The TAP Display Adapter is a lightweight process, written in C++ on the Qt framework, that acts as an IPC proxy between the TAP Engine and the TAP Display. It receives data from the TAP Engine and forwards it to the TAP Display. It also records the data received from TAP Engine in the event it is needed to synchronize a connected TAP Display with the rest of the system. TAP incorporates limited use of gestures, which are abstracted through the Qt framework, as is switching of the display configuration depending on screen orientation. The UTAS TIM appliance is a certified power source which was utilized for FT-2 and will be used to provide power to the tablets for the Alaska Airlines operational trial. Fig. 2 shows a screenshot from the iOS tablet application.

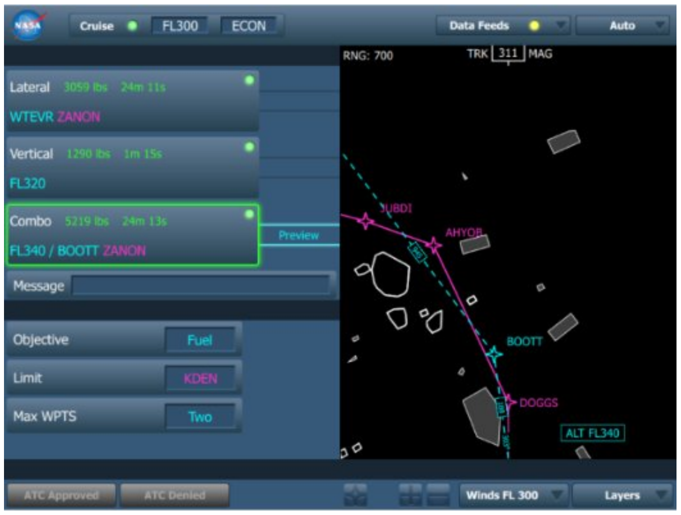

Fig. 2. TAP Display auto mode example

V. Networking

Onboard local networking services are increasing in availability and capability, but significant variability exists from airline to airline and even among different classes of aircraft comprising an airline's fleet. The availability of wireless technologies is also increasing but cannot be assumed to be available. Where it is available, there are often restrictions to its use. Wide-area networking services, such as broadband Internet is also growing in adoption in the cockpit [6]. EFB software developers must balance the use of Internet data sources and the cost of inflight broadband bandwidth, which can discourage budget-conscious users from adopting an application.

For the Alaska Airlines operational trial, wireless communications may or may not be available. An Ethernet network will be available, but the iPad does not support Ethernet communication, leaving only Universal Serial Bus (USB) communications. The UTAS TIM hardware appliance will host the TAP Display Adapter. It is an ARM-based architecture running a custom CentOS Linux kernel. A set of proprietary UTAS services are installed on the device to facilitate communication between the tablet, TIM, and UTAS AID. A Software Development Kit (SDK) and an iOS framework is also provided to integrate with iOS apps for communication with the TIM over USB, Bluetooth Personal Area Network, and eventually WIFI. Both the TAP Display Adapter and TAP Display were developed with a modular IPC approach and an abstract communications interface. This allows for the IPC to be swapped out as needed without requiring rewrite of existing functional code. It also allows for mixed IPC within the same application. For example, the Alaska Airlines implementation of the TAP Display Adapter, running on the TIM, will utilize TCP/IP (Transmission Control Protocol/Internet Protocol) over an Ethernet connection for IPC with the TAP Engine and USB IPC for the TAP Display.

VI. Resource Management

Hardware manufacturers are positioning their products to allow hosting of third-party applications in the cockpit. In order to provide a consistent quality of service for all installed applications, these platforms can impose limits as to the amount of memory, processor time, and storage capacity an EFB application can use. The imposed limits can either be soft guidelines for developers to follow or hard limits enforced by the hosting platform itself.

As a derivative of AOP, TAP was not specifically designed with these considerations in mind, and computing resources were not a restriction in the Piaggio Avanti flight-test aircraft. In the planning phase for the partner airline operational trials, it became necessary to evaluate the resource footprint of TAP and implement measures to control them. Active control of processor and memory utilization is not possible without a redesign of the TAP suite, including legacy AOP code, which was not an option. Therefore, development focus was placed on limiting its static storage footprint. UTAS has specified a limit for static storage of each hosted component. To satisfy these limits, a mechanism has been developed to reduce the footprint of run-time diagnostic output in stages, as the project moves from testing to full operation. Additionally, file management processes have been implemented to compress and offload run-time output to a central repository at application shutdown for later retrieval. A scavenging process has been implemented to ensure that data left behind, due to improper shutdown from a previous flight, is properly compressed and offloaded when the application is restarted. In the unlikely event that these processes fail to keep the static storage from overrunning the allotted space, the operating environment is evaluated at startup, and TAP will be prohibited from execution until the problem can be rectified. The TAP Service daemon is responsible for these tasks.

Qualification of processing and memory resources will rely on extensive testing on the host hardware. For Alaska Airlines, the TAP Engine is expected to be hosted on the UTAS AID. UTAS and NASA will be conducting tests to determine whether the TAP Engine can perform sufficiently for the trial, without adversely affecting the performance of the ARINC 834 server or other software processes on the AID. The onboard Gogo ACPU-2 unit will also be evaluated as hosting platform for the TAP Engine and is being evaluated for performance as well. EDS is expected to be hosted on the Gogo ACPU-2 unit. Testing is also underway to evaluate the TAP Display Adapter on the UTAS TIM host appliance. As an objective of FT-2, the performance of TAP Display on the iOS tablet platform has been determined to be acceptable. Resource limits have yet to be established for the Virgin America operational trial.

VII. Avionics Data

EFBs receive avionics data via the ARINC 834 server, supplying data in the Simple Text Avionics Protocol (STAP). The server may be hosted on an installed device such as an AID. Avionics hardware is hard-wired to input ports on the AID. Applications subscribe to ARINC 429 word labels on individual ports of the AID via the ARINC 834 server. The ports and word labels subscribed to vary between installations due to differences in installed hardware that is wired to the AID. Any changes to the AID wiring to support EFB software may require the system to be re-certified, delaying deployment and incurring additional cost. In early planning, it was determined that Alaska Airlines and Virgin America had no preexisting provisions for receiving active route (i.e. flight plan) data into their EFB systems. For Alaska Airlines, the data is available from their Flight Management System (FMS) through a dedicated intent output bus, but it had not been wired into the AID. This change requires a Supplemental Type Certificate (STC) for the aircraft. The FMS employed by Virgin America does not provide an intent output bus. Active route data is available on the Electronic Instruments System (EIS) bus, however making this data available to the installed NEXIS EFB also requires an STC.

Existing applications on installed EFBs or tablets integrated with an installed AID can access aircraft state data including, but not limited to, current position and altitude. Obtaining, decoding, and translating aircraft route data and Automatic Dependent Surveillance Broadcast (ADS-B) IN traffic data represents new ground for EFB software, and this presented one of the biggest challenges for developing and deploying TAP. In order to calculate fuel and time savings of route optimization solutions, TAP needs to be able to accurately predict the aircraft's entire flight trajectory, which requires data that is not readily available in route data conforming to ARINC 702A-1 protocol. The protocol also does not include waypoint alphanumeric names that TAP uses for display of the current route on the HMI. For FT-1, the Piaggio Avanti flight-test aircraft provided route data in the General Aviation Manufacturer's Association (GAMA) format. For FT-2, the same GAMA route data was utilized, but without destination, waypoint names, cruise altitude, speed, and waypoint constraints, to simulate the ARINC 702A-1 route format used by Alaska Airlines.

To support ARINC 702A-1 route format, the TAP system had to provide functionality to allow users to enter the missing data. To make this process as intuitive as possible, methods were developed to provide the waypoint names and to monitor aircraft state data to determine if the pilot-entered cruise altitude and speed data are consistent. Upon receiving an initial route, the TAP Engine provides its best guess as to the names of the waypoints via database lookup based on spatial proximity to the supplied latitude and longitude values, along with a list of alternate waypoint names for each. These data are provided to the TAP Display, where the pilot examines and confirms the waypoint names on the initial TAP Display Startup Checklist page. This page also provides an interface for the crew to input the other required flight information that is not available via ARINC 702A-1 (i.e., cruise altitude, cruise speed, and destination). The TAP Display also provides another tool, the Performance/Constraints page, to allow users to enter/edit these data types to account for changes in the Startup Checklist entries after initial operation.

Virgin America's FMS has no available outut bus for active route (i.e. FMS flight plan) data. Instead, the only available intent information is provided on the EIS bus in support of the on board navigation display. Since the transmitted route data is dependent on the current scaling factor of the navigation display, pilots will be required to "walk" through the route on the navigation display prior to the flight, and the TAP Engine will compile the data until the full route is obtained. Manual changes to the route such as reroutes issued by ATC, execution of a TAP advisory, or other changes, will require the pilot to repeat this process to allow the TAP Engine to update its own knowledge of the aircraft's active route.

ADS-B data is similarly limited because it too is currently geared toward cockpit displays and not general use. ADS-B receivers generally produce traffic data in the Display Traffic Information Files (DTIF) format. This format represents preprocessed ADS-B for consumption by on board Cockpit Display of Traffic Information (CDTI) devices. The Aviation Communication & Surveillance Systems (ACSS) TCAS (Traffic Alert and Collision Avoidance System) 3000SP hardware aboard the Avanti Piaggio flight-test aircraft does not transmit traffic data without a CDTI connection. For FT-1 and FT-2, the connection had to be spoofed in order to allow traffic data to be available through the AID. This was possible onboard the test aircraft, but would likely require an alternate solution or additional STC for a commercial airline. Alaska Airlines will use the same ADS-B hardware as the Avanti test aircraft. Virgin America will be using the Honeywell TPA-100C.

The TAP Engine reuses the legacy AOP traffic processing subsystem. It is based on an experimental implementation of the ADS-B Minimum Aviation System Performance Standards for NextGen concepts, classifying ADS-B data in terms of "reports" such as State-Vector Reports (SVRs), Target State Reports, and Trajectory Change Reports. The ADS-B data available on aircraft today most closely relates to SVRs. The TAP Engine avionics I/O decomposes DTIF data and produces internal SVRs. Since DTIF data is essentially traffic state data, no strategic intent is known for surrounding aircraft, limiting TAP Engine's traffic predictive ability to a time-extrapolated vector of the current traffic aircraft state. Future plans to integrate with the FAA's System Wide Information Management system may provide full traffic intent data to integrate with on board avionics, facilitating greater predictability of surrounding aircraft.

VIII. Adaptation of Avionics Data

For FT-1, FT-2, and the upcoming partner airline operational trials, a detailed analysis was performed for each environment to identify the available data, source, precision, and frequency of data transmission. From there, an adaptation path was determined to provide TAP Engine with the proper native data structures it needs to operate. Adaptation is complicated due to the varying formats, limitations, and sources of data. It was desirable for TAP to be able to handle differences from platform to platform without requiring wholesale changes to the functional code. As a research support platform, AOP was designed to be modular, allowing new concepts to be tested based on functional objective (i.e., hazards processing, trajectory generation, conflict detection, conflict resolution, crew I/O, and avionics I/O). At a high level, implementing the TAP Engine consisted of swapping or removing some of these functional units and replacing them with those written specifically in support of TASAR. To allow maximum reuse of existing AOP code, the avionics I/O component was replaced with a STAP compatible I/O, which reads STAP data and provides compatible AOP data to the rest of the system. The crew I/O module was replaced with a version that is compatible with the TAP Display. Since TAP is not a separation assurance system, the conflict detection and resolution components were replaced with a new optimization component, which combines elements of both but with a different functional purpose, i.e., to qualify proposed route optimizations as conflict-free before presenting them to the crew.

The TAP Engine avionics I/O component uses an environment-specific subscription configuration object. The configuration object manages the ARINC 834 subscription details per operating platform and assigns them to abstract functional designations (e.g., route data, state data, guidance data, wind data, traffic data). The higher-level I/O logic dispatches the received data, according to its functional designation, to abstract handlers, allowing the higher-level I/O code to be mostly unchanged for any environment it is operating within. The specific decoding of ARINC words and the aggregation of decoded data into native internal structures is done via abstract I/O handlers, with derived implementations for each supported environment. This design allows for greater flexibility for currently planned and future TAP implementations, while reducing development and testing time.

IX. Internet Data

Inflight broadband technologies are expanding, allowing EFB applications access to data not normally available on the aircraft. [6] TAP was developed to utilize broadband via the EDS. EDS connects to Internet sources for winds, convective weather, and SUA schedules and serves them to one or more instances of the TAP Engine running on the aircraft. These data are then integrated with onboard avionics data. Because the cost of broadband data usage can be high, EDS was designed to minimize consumed bandwidth to the extent possible while maintaining operational requirements.

X. Deployment and Operational Logistics

To facilitate installation of the TAP software suite to the onboard hosts, installation scripts are used. The scripts are designed to allow aircraft maintenance crews to install the application suite with a minimum of effort. The installation packages are customized for the aircraft, including the binary executables, configuration files, aircraft performance data, network addresses, and target host locations. Using a computer connected to the aircraft's internal network, the maintenance personnel executes the script to install the preconfigured software suite. The TAP Utility is used to validate the installation. The TAP Utility can be run on Windows, Linux, or iOS. Each onboard computing platform hosting one or more TAP components has an associated TAP Service daemon running on the host. The TAP Utility communicates with the TAP Service daemons to validate the installation. If there is a problem, the TAP Utility provides information to the user, as to which components have failed to install, to aid in troubleshooting.

When the aircraft is first powered up, the host hardware is configured to launch the TAP Service, which will wait for a signal to launch its hosted TAP components. This is done to avoid having TAP launch itself reflexively, using resources unnecessarily, and possibly incurring cost for broadband data when the system is not being used.

For the Alaska Airlines operational trial, there will be two UTAS TIM appliances in the cockpit for the Captain and First Officer, respectively. Upon connecting to the TIM and launching TAP Display on the iOS tablet, the TAP Display will signal the TAP Service daemons to launch the rest of the system. Each seat will connect to its own instance of the TAP Engine. Both seats will communicate to a single EDS host for broadband data.

During flight, the TAP Utility, which is installed on the EFB, provides an interface to the installed TAP components in the event that the system becomes unresponsive. The user can bring the system down and relaunch it to resolve most problems in flight. This is necessary because the crew has no other interface to the distributed TAP components.

At the conclusion of the flight, the TAP Service is responsible for compressing and offloading output data from all TAP components to a central repository for offline retrieval and evaluation. For the Alaska Airlines operational trial, this is complicated by the fact that the iOS device stores its output data in the application bundle on the device. It would be difficult to collect that data in a cockpit setting. The absence of standard network access precludes the use of services such as syslog for remote logging. Instead, the TAP Display output data will be written to the UTAS TIM appliance via the UTAS SDK over USB. The TAP Display output data will be stored on the TIM and pushed to the central repository by the TAP Service running there as needed.

XI. Conclusion

Advanced EFB applications such as TAP have the potential to greatly expand the capabilities of cockpit operation at reduced installation, maintenance, and operating costs if the right hardware, data, and communications services are available. For both hardware manufacturers and commercial airlines, the example of TAP presents the current technical and operational hurdles that need to be addressed when developing, certifying, purchasing, and deploying new cockpit hardware. Avionics hardware manufacturers need to consider the future requirements of EFB software when designing their products to provide expanded and standardized access to on board avionics data, as well as high-performance computing platforms and networking services. Commercial airlines need to consider choosing platforms that are best positioned to facilitate deployment of next generation EFB applications, in the shortest time and lowest cost possible. Finally, EFB software developers need to consider the variables and obstacles presented, when designing advanced applications, in order to provide a product with the functionality and flexibility necessary to appeal to a wide market, while reducing development time and cost.

References

[1] Ballin, M.G. and Wing, D.J., “Traffic Aware Strategic Aircrew Requests (TASAR)”, AIAA-2012-5623, *AIAA 12th Aircraft Technology, Integration, and Operations Conference (ATIO)*, Indianapolis, IN, USA September 2012.

[2] Henderson, J., “Traffic Aware Strategic Aircrew Requests (TASAR) Concept of Operations”, NASA/ CR-2013-218001, May 2013.

[3] Woods, S.E., Vivona, R.A., Henderson, J., Wing, D.J. and Burke, K.A., “Traffic Aware Planner for Cockpit-based Trajectory Optimization”, accepted to AIAA 16th Aviation Technology, Integration, and Operations (ATIO) Conference, Washington, DC, USA, June 2016.

[4] Karr, D A.; Vivona, R A.; Roscoe, D A.; DePascale, S M.; and Wing, D J.: “Autonomous Operations Planner: A Flexible Platform for Research in Flight-Deck Support for Airborne Self-Separation.” AIAA-2012-5417, Sept. 2012.

[5] Maris, J. M., Haynes, M. A., Wing, D.J., Burke, K.A., Henderson, J., and Woods, S.E., “Traffic Aware Planner (TAP) Flight Evaluation”, AIAA-2014-2166, 14th Aircraft Technology, Integration, and Operations Conference (ATIO), Atlanta, GA, USA, June 2014.

[6] Gogo LLC: “From the Ground Up: How the Internet of Things will Give Rise to Connected Aviation.” Published by Gogo LLC, 2016.