Capability Driven Architecture: An Approach to Airworthy Reusable Software

Richard Zepeda
Software Engineer

Stephen Simi
Project Manager

Dennis Kenman
Program Manager

Sean Mulholland
System Architect

Elden Crom
Software Engineer

Virgil Swadley
Software Engineer

Tucson Embedded Systems, Inc.
Tucson Arizona
dennisk@tucsonembedded.com

Abstract

Current and projected program requirements are exceeding Department of Defense (DoD) budget and schedule constraints. This applies to the Army's requirements to integrate common avionics equipment onto dissimilar rotorcraft – both manned and unmanned. As such, innovative approaches are needed to address the integration costs and time. The Common Software Initiative (CSI) was formed by the U. S. Army's Product Manager of Aviation Mission Equipment (AME) to explore solutions for this problem. In support of CSI, Capability Driven Architecture (CDA) has been architected and demonstrated to AME as an architecture designed for reuse. It is an open-standards based architecture for integrating and deploying new and legacy capabilities and avionics onto Army rotorcraft. The planned goal for CDA is 100% reuse, such that one piece of software may be certified and reused across multiple platforms as described in the FAA circular AC 20-148 [1]. The CDA architecture can be applied to all capabilities including communications, navigational, sensors, actuators, etc., and, as a proof of concept, it was first developed and demonstrated for radio control as CDA Radio Control (CDA-RC).

Introduction

The Army's Product Manager, Aviation Mission Equipment (PM–AME), is seeking to implement a process by which common software products, to include common avionics integration software, can be identified, acquired, tested, and integrated across the disparate Army Aviation platforms.

PM-AME has identified the need for this process through the Common Software Initiative (CSI). Implementation of the CSI would position AME into conformance with the acquisition strategy outlined in Chapter 2 of the Defense Acquisition Guidebook [2] and with the directives of AR 70-1 Army Acquisition Policy [3]. These two DoD documents outline prescribed requirements for standardization, commonality, and systematic reusability that will guide Army Aviation practices for improving budget-to-capability performance.

In support of CSI, Capability Driven Architecture (CDA) has been demonstrated to AME as an airworthy design for creating reusable software components. It is an open architecture for integrating and deploying new and legacy capabilities and avionics onto Army rotorcraft. While architectures exist that can claim software reuse, few, if any, can claim software reuse for safety critical airworthy applications.

The planned goal for CDA is 100% reuse, such that one piece of software may be developed, tested, and certified then reused across multiple disparate platforms as described in the FAA circular AC 20-148 – Reusable Software Components [1].

Background

The Army has an ongoing need to integrate Aviation Mission Equipment products into aviation platforms. This integration can occur at aircraft delivery or as an aircraft upgrade. The integration cycle includes a significant effort in developing software to interface to new and changing AME Products.

Each platform prime contractor is responsible for developing the software to interface with new aviation equipment. Historically, equipment was introduced as mission-specific, and added as non-integrated ("strap-on") equipment into their respective platforms.

Today's aviation mission equipment is highly integrated into the platform and moreover the same equipment is integrated within different platforms.

This arrangement has lead to ad hoc development and stovepipe systems resulting in duplication of effort across the aviation platforms for integrating common aviation equipment. It has also resulted in duplication of efforts within an aviation platform when integrating a new piece of

Presented at the American Helicopter Society 63rd Annual Forum, Virginia Beach, VA, May 1-3, 2007.

aviation equipment that has similar functional capabilities to already integrated equipment.

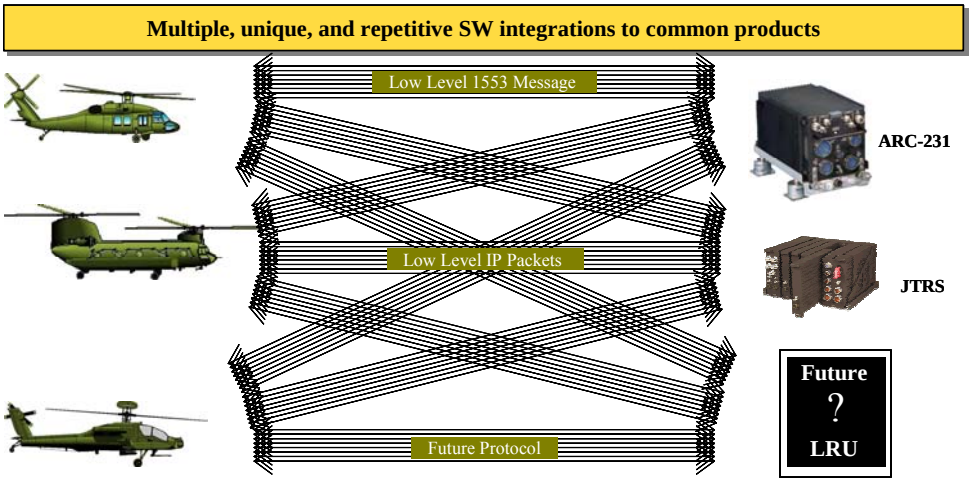
The result is that current and projected program requirements are exceeding budget and schedule constraints. To address these issues, both technological and process solutions must be developed within the Aviation community. Technological solutions must be based on the integration of functional capabilities across aircraft, and process solutions are needed to accommodate cross-platform integration and certification requirements.

What follows is an overview of an architecture design along with process suggestions that will allow such an architecture to be verified, certified, and reused across the aviation fleet. A description of the development of the architecture for radio control is included along with a summary of the demonstrations that explored the viability of common, reusable software within Army Aviation.

Motivation for Architectural Transformation

The motivation behind the Capability Driven Architecture design is to provide a common interface to a category of similar devices, much like desktop computer applications have a common interface to the myriad of computer printers and other peripherals. Currently, aviation applications have nothing similar for integrating avionics equipment.

What does exist is a mix of disparate aviation platform architectures and stovepipe programs based on proprietary interfaces. Illustrated in Figure 1, Source of Problem, is the implementation of Aviation Mission Equipment on Aviation's rotary aircraft fleet. For every Line Replaceable Unit (LRU) update or change, implementation-specific changes are required on each and every aircraft.



Each platform:

- Develops *"unique"* software to integrate Avionic products at the low-level interfaces.
- A-Kit development is tightly coupled to *detailed* B-Kit interface definition.
- Maintains expertise on the low-level interfaces to each Avionic product.

Figure 1. Source of Problem

The personal computing (PC) industry had a similar problem and devised its solution decades ago. The solution is architectures based on standardizing (making common) its interfaces. They separated the use of a capability (e.g., File→Print) from its implementation (e.g., bit-level instructions to a laser printer versus a dot matrix printer, etc). The Aviation parallel for Communications is the Set→Frequency command for their ARC-201D, ARC-231, etc., radios.

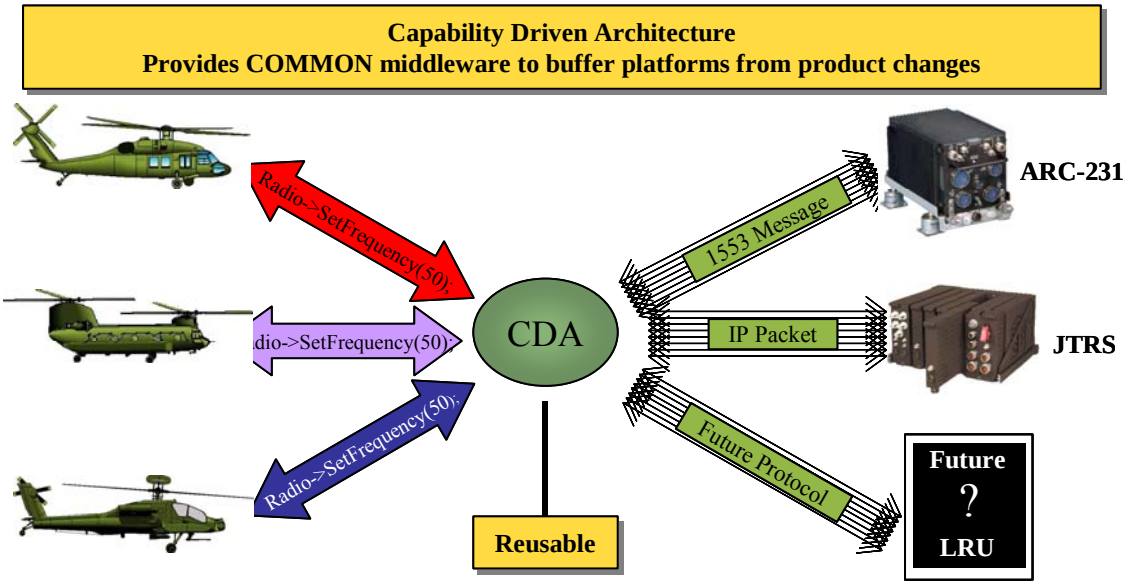
Therefore, an architecture was envisioned that would be non-platform and non-LRU specific implementations. It should enable reuse through abstraction and extensibility, and be architected to reduce the time and effort associated with integrating common equipment across various dissimilar platforms. The Capability Driven Architecture (CDA) is one such approach.

The CDA approach emphasizes the integration of capabilities rather than integration of the specific systems, sub-systems, or hardware. It should apply to integrating grouped capability sets, such as those used by communications equipment, navigational aids, sensors, actuators, etc. As a proof of concept, CDA was first developed and a prototype demonstrated for radio control as CDA-RC.

To date, there have been two successful demonstrations of software reuse using CDA-RC on three different platforms. The first demonstration occurred at the U.S. Army Technology Integration Center. The second demonstration occurred at the Aviation Systems Integration Facility (ASIF).

The first demonstration implemented the radio control software for two tactical radios (ARC-210D and ARC-231) for one platform. The second demonstration implemented a subset of the same radio control software for one tactical radio onto two airworthy platforms.

Illustrated in Figure 2 is the conceptual "Solution" – an architecture based on a set of common interface standards and common middleware control code or translation software to buffer the target platforms from product changes. The specifics of this architecture, Capability Driven Architecture, are described below.



- Develop integration software that is common (not "unique") across platforms.
- Integrate the capability, not the product; for example, provide identical interfaces to SINCGARS satisfied by ARC-201, ARC 231, and JTRS.
  - Remove the platform's burden of implementing low-level interfaces to Avionics products.
  - Platforms are insulated from Avionics product changes not affecting functionality.
  - De-couples A-Kit/B-Kit development cycle.

Figure 2. The Solution

Capability Driven Architecture Overview

Abstraction

The key to common, reusable software is being able to isolate the software from the differences between the various platforms and avionics. By isolating device control software from differences between platforms, the software becomes platform independent and reusable across the platforms. By isolating application software from differences between avionic devices, the platform applications become device independent and reusable with different devices.

For these reasons, CDA replaces platform-unique integration code with two layers of abstraction, an operating environment (OE) abstraction layer and a device abstraction layer (see Figure 3).

Through these two layers of abstraction, CDA effectively reduces the duplication of integration efforts in two ways, across the platforms and within a platform. Firstly, the integration efforts across platforms are reduced since the software for controlling common avionic equipment is platform independent and can be used by all platforms. Secondly, the integration efforts within a platform are reduced since virtually an entire category of devices is integrated into a platform by using a single interface. This means that different devices in the same category become largely interchangeable so that the addition of a new device or swapping with a similar device having similar functionality requires little effort.

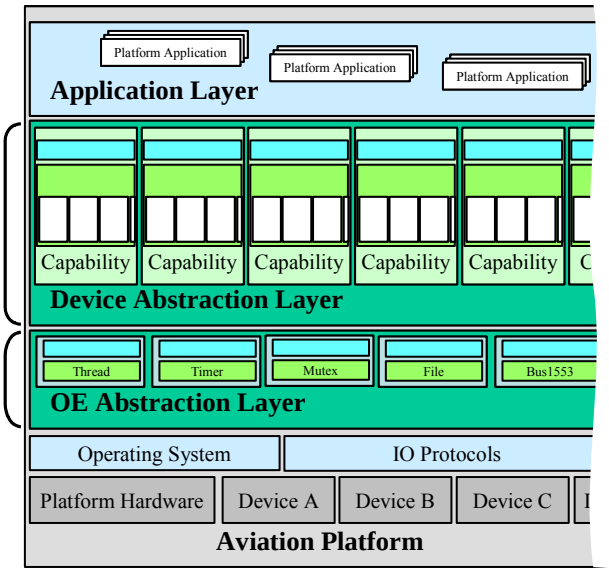


Figure 3. Aviation Platform and CDA Layer Overview

Operating Environment Abstraction Layer The operating environment (OE) consists of a platform's hardware and computer operating system − essentially those parts of a system that define the platform to a CDA implementation. To remain platform independent, the implementation code does not communicate directly with the operating system or hardware. Instead, the implementation accesses OS services and other protocols, such as threads and timer services or IO protocols, through the interfaces defined in the OE abstraction layer.

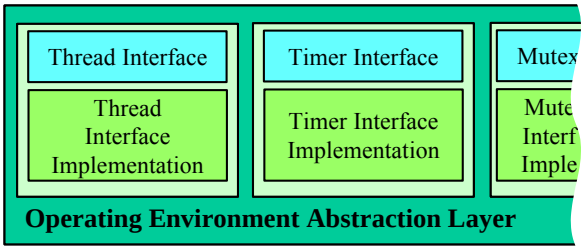


Figure 4. The OE Abstraction Layer

Device Abstraction Layer The device abstraction layer provides the interface to the user application. It contains the bulk of the CDA implementation and allows platform application code to be independent of the devices integrated on a platform. This abstraction layer is where actual integration of devices takes place.

To perform this abstraction, a category of similar devices is broken down into its core Capabilities. A Capability is a collection of functions with a related purpose. For example the functions that change the volume and squelch belong to the Voice Capability. Within in the category of radios, the functions that change the frequency and output power belong in the general Radio Capability. The radio category also has many other Capabilities such as Message, SATCOM (satellite communications), Test (for built-in tests), and Crypto (for COMSEC functions) to name a few.

At the top level of a Capability is the capability driven interface (see Figure 5). This interface defines the API used by platform applications to access Capability functions regardless of the device being controlled. This enables integrators to shift from the convention of integrating devices (device-centric) to a practice of integrating Capabilities (Capability-centric). Thus, the code for a well-designed application using a particular Capability will not need to change when replacing one device with a different device that has the same Capability [e.g., replacing an ARC-201 implementation of Single Channel Ground and Airborne Radio System (SINCGARS) with a Joint Tactical Radio System (JTRS) implementation of SINCGARS]. Yet, Capability Driven Architecture still allows a device-centric approach for an application by regarding a collection of Capabilities as a particular device. In other words, the application can be implemented using grouped Capabilities such that each group is treated as a device. This technique can allow current applications to use CDA with minimal modification, albeit the interchangeability of like devices may be limited.

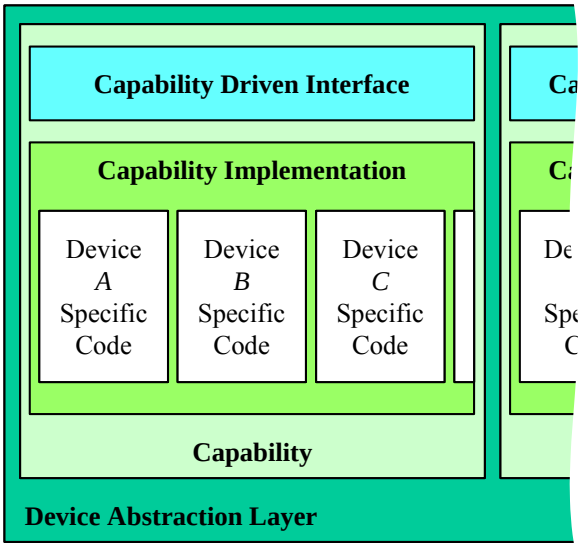


Figure 5. The Device Abstraction Layer

Below the capability driven interface is the Capability implementation. The implementation contains the code that is specific to the various devices and is hidden from the platform applications by the capability interface.

A Concrete Exercise – CDA-RC

Currently, there have been two successful CDA-RC demonstrations through the AME Common Software Initiative. The first CDA-RC demonstration integrated the full set of functionality of two radios, the ARC-201D and the ARC-231, onto the Army Aviation Systems Integration Facility's Aviation Test and Integration Center (ATIC) platform. Control for both radios was implemented using one common interface. The second demonstration integrated the same CDA-RC software on two disparate aviation platforms, the MCAP II and the CAAS platforms, and successfully controlled a subset of functionality of an ARC-201D on these two platforms.

The demonstrations of Capability Driven Architecture for radio control were a combined exercise in rapid development and integration of common aviation software culminating with two demonstrations. The U.S. Army's Product Manager of Aviation Mission Equipment and the Aviation Applied Technology Directorate sponsored the demonstrations.

The demonstrations were concerted efforts of TES (Tucson Embedded Systems), the Apache Integrator, the Chinook Integrator, and the ATIC Integrator. TES served as a third-party developer of common software

Development

The development environment consisted of a Linux PC with a Condor QPCI 1553 interface card, two ARC-201D radios, and two ARC-231 radios.

The process of abstraction constituted defining the radio control requirements. To begin, TES analyzed the interface control documents of five LRUs, abstracting the specific radio functions into common function calls belonging to the CDA-RC API (see Figure 6). (In addition to the ARC-201D and ARC-231, several legacy radios were used to obtain a more generalized abstraction.) This iterative process resulted in defining the radio control Capabilities by mapping data fields of MIL-STD-1553 messages to function parameters and by review of functional relationships and radio commonalities.

To illustrate mapping specific radio functions into a common API, both an ARC-201D and an ARC-231 are capable of changing the radio frequency on a single channel. However, the MIL-STD-1553 message used by the ARC-201D represents frequency with four fields (the tens, ones, tenths, and hundredths digits of frequency expressed in megahertz). The message used by the ARC-231 represents frequency in a single field expressed in kilohertz as a 32-bit integer. These fields were mapped to a single parameter of a function called SetFrequency. The SetFrequency function belonged to the Radio Capability. The Radio Capability contained other general radio related functions such as SetPower for changing the transmit output power.

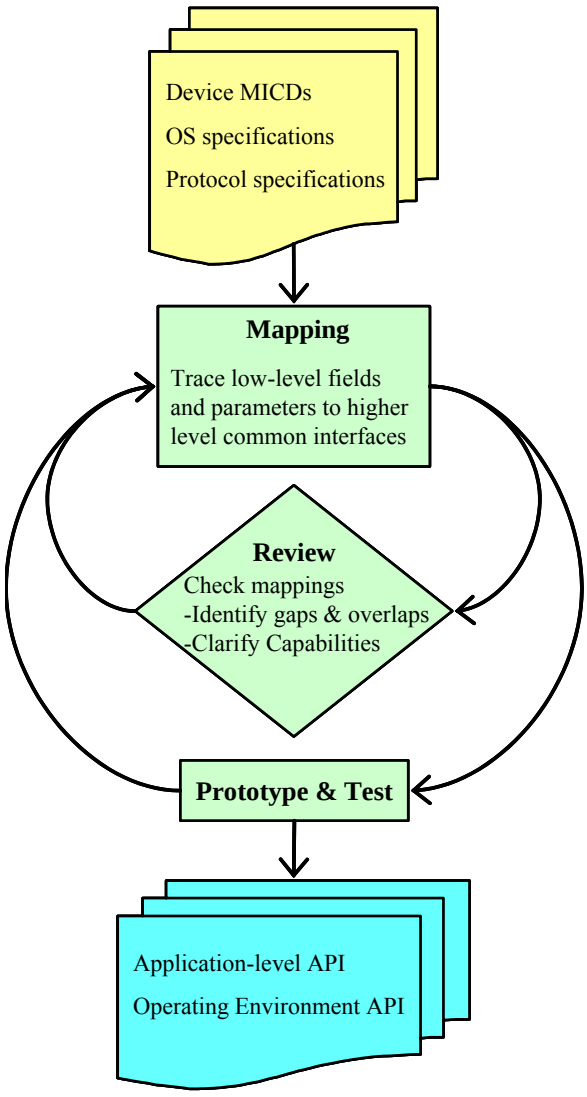


Figure 6. Iterative Abstration Process

Within the CDA-RC API, one radio is represented by many Capabilities. Though all the radios are different, Table 1 shows their capabilities can be abstracted such that the overlap is sufficient to justify a common interface, and Figure 7 further illustrates how the process reduces documentation used for integration from many to few.

Table 1. Abstracted Capabilities for Radio Control

Capability	ARC-201D	ARC-231
Channel (handles presets)	x	x
Crypto	x	x
DAMA		x
HaveQuick		x
Message	x	x
Modem	x	x
Radio	x	x
SatCom		x
SINCGARS	x	x
Test (manages BITE)	x	x
Voice	x	x

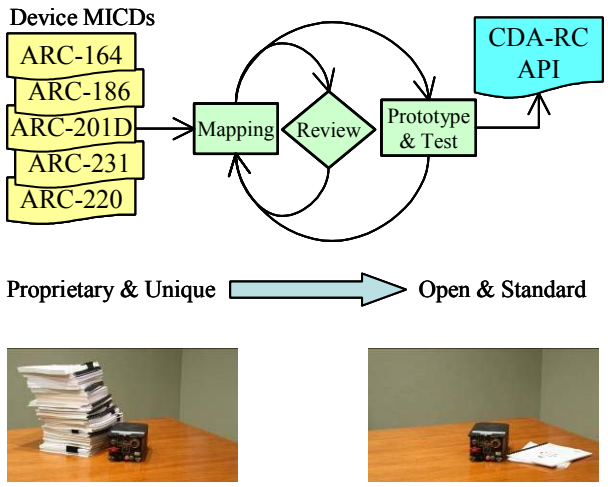


Figure 7. CDA Reduces Many MICDs to One

A similar abstraction process was followed to develop an operating environment (OE) API. The operating environment API allowed CDA-RC implementations to use operating system (OS) services such as threads, mutual exclusion, file operations, and timers; yet it isolated the implementations from the specifics of every operating system on which CDA-RC ran.

The OE API also isolated CDA-RC from the specifics of sending and receiving messages to a radio LRU via the Bus1553 message set. In the development environment, TES implemented the OE abstraction to send messages directly to the MIL-STD-1553 bus.

In the demonstration environments, the integrators implemented this portion of the OE abstraction to send messages to an IO handler via internet protocol. Moreover, since TES was unable to test in the target environments, the whole of the OE abstraction implementation was to be open and the source was delivered to the integrators in case any platform-specific alterations were needed.

Integration and Demonstrations

Commonality Working Group The U.S. Army's Product Manager, Aviation Mission Equipment (PM-AME) and the Aviation Applied Technology Directorate (AATD) co-sponsored the Commonality Working Group common software demonstrations. The efforts were an experiment of how OEM platform integrators and third-party developers can come together to integrate avionics equipment with common software.

This effort examined the processes, documentation, and implementation of integrating common reusable control code of an ARC-201D radio onto Army Aviation platforms using a subset of CDA-RC functions.

The integration environments of CDA-RC for the Commonality Working Group involved two disparate platform architectures. One OEM integrated and demonstrated CDA-RC onto the MCAP II (Manned/unmanned Common Architecture Program phase 2) architecture–the architecture to be used in the future AH-64D Apache. Another OEM integrated and demonstrated onto the CAAS (Common Aviation Architecture System) architecture–the architecture is used in the MH-47 and MH-60 helicopters (and scheduled for use in the CH-47 Chinook, UH-60 Blackhawk, and the ARH-70 helicopters).

Tucson Embedded Systems (TES) worked with AME's OEMs to assist their integration efforts of the CDA-RC software on the MCAP-II and CAAS platforms respectively.

The integration effort included:

- Initializing the ARC-201D for single channel tuning and voice only,
- Providing interface to (integrating to an existing MCAP-II and CAAS HMI or human-machine interface) and demonstrating single channel tuning (volume, power, squelch, frequency), and
- Providing documentation that describes the methods and strategies used to mask and mitigate the OSA differences (MCAP-II and CAAS) in order to facilitate the use of the common software across platforms.

The specific scope of the integration effort included the entire operating environment API, but only a subset of the application-level API. The following lists the specific functions that were integrated during the CWG Common Software demonstration.

Application Level API (subset):

Voice::SetVolume	Voice::GetVolume
Voice::SetSquelch	Voice::GetSquelch
Radio::SetFrequency	Radio::GetFrequency
Radio::SetPower	Radio::GetPower

Operating Environment API:

Thread::SpawnThread	Thread::WaitForThread
Thread::ProcessThread	Thread::StopThread
Thread::KillThread	Thread::IsActiveThread
Thread::SetDelay	Thread::Delay
Thread::IsTerminatedThread	
Thread::IsFinishedExecution	
Thread::GetSpawnedThreadID	
Thread::GetCurrentThread	

File::Open	File::Close
File::Write	File::Read
File::ReadLine	File::IsOpened
File::Flush	

Timeout::Delay	Timeout::DelayDifference
Timeout::End	Timeout::GetInterval
Timeout::Slide	Timeout::Start
Timeout::SystemTimeStamp	

Mutex::Lock	Mutex::TryLock
Mutex::UnLock	

Bus1553::Init	Bus1553::Receive
Bus1553::Send	

Army Technology Integration Center The purpose of the Army Technology Integration Center (ATIC) program was to establish an open architecture, system of systems, airframe-independent, system integration test facility. The ATIC was to increase performance and commonality, and reduce the cost of helicopter mission equipment packages by facilitating rapid integration and evaluation. During its development and construction, the ATIC Integrator looked to CDA-RC to provide radio control for their phase I demonstration of the ATIC's abilities.

For the Phase I demonstration, the ATIC Integrator integrated CDA-RC into a VxWorks environment. While demonstrating ATIC's reconfigurable abilities, the demonstration also presented the advantages of the CDA device abstraction layer which enable the changing of devices without changing the code of the applications that use them.

The demonstration started by exercising all the functionality of the ARC-201D radio through the CDA-RC Capabilities with the ATIC HMI. Then using the same HMI without changes, the Capabilities that were demonstrated on the ARC-201D were also demonstrated on the ARC-231 along with the ARC-231's additional functionality.

Lessons Learned

The integration efforts of the CDA-RC for the Commonality Working Group (CWG) proved insightful. Lessons learned document [4,5] identified programmatic, technical, and process related issues.

Seven lessons were identified: (1) Understand the target platform(s), (2) Obtain a capable common software repository, (3) Control the configuration, (4) Make the documentation more useful for integration, (5) Address programmatic licensing issues early, (6) Make allowances for the newly-defined developer-integrator process, and (7) Understand that open systems architectures are not entirely open.

These lessons are described along with suggestions as enumerated.

1) Understand the Target Platform(s) – Although there were several technical exchange meetings between the common software developer and the integrating OEMs, original assumptions of target architectures were incorrect which resulted in time and effort building and testing against incorrect target platforms. A Government-owned Aviation Systems Integration Facility (ASIF) should be developed to support the common software development process. The ASIF should be loaded with target architecture build suites along with mobile build, development, and integration labs. These build suites need to be available to all participants, both Developers and OEMs, to assist with the development, integration, and validation of future common software efforts. Another white paper [6] describes such a development environment.

2) Obtain a Capable Common Software Repository – The Army Knowledge Online (AKO) system was used as the central shared repository. This repository both worked and had capability shortcomings. While the repository provided an integration-neutral area to share files and information, as a tool, the AKO was unfriendly, had poor access control features, and was difficult to upload batches or sets of files from multiple directories. Therefore, suggested was to use a tool other than the Army Knowledge Online (AKO) system. The topic and issue of Common Software Licenses is described below in (5).

3) Control the Configuration – common software files should be better managed and configurations synchronized. An issue occurred when the Integrators linked to non-synchronized software. This issue surfaced when the Developer modified implementations of header files to assist (we tried to simplify) integration efforts (i.e., TES moved #defines for OE specifics) and these files were not re-synchronized causing Integrator builds issues. Participants should weigh and accept the trade-off of being in a real-time prototype rapid-development environment versus a more-formal production environment. In a more-formal

environment, configuration management (CM) issues are better controlled, but at a cost of extra time and resources. The CWG accepted the risk of working within the less-formal proof-of-concept rapid-development environment in order to expedite the prototype process and meet its short demonstration schedule.

4) Make the documentation more useful for integration – The original interface control document (ICD) provided to the Integrators was not adequate as a Developer/Integrator's User Guide. Participants should establish and maintain an "open dialog." This was critical to address questions so that integration efforts could continue to move forward. On-site visits and open communications were essential for complex integration efforts to succeed. The CWG actually did very well here and forged relationships that will serve and assist with AME's future goals for the CSI.

5) Address programmatic licensing issues early – Proprietary markers on software files and licensing issues hampered initial file sharing among participants. A part of this study was to determine the *How*, that is how to develop common software from both programmatic and technical standpoints. The FAA AC 20-148 Reusable Software Components [1] identifies that these types of undertakings require considerable up-front planning and suggest allotting time on "Stakeholders Agreement" and defining the communication channels and roles among stakeholders. We simply underestimated the need to address business interests from the corporate level.

6) Make allowances for newly defined developer-to-integrator processes – The CWG agreed to conduct a proof of concept and rapid prototype development demonstrating that common software can be used on disparate platforms. To accomplish the effort, engineer-to-engineer interfaces and exchanges occurring in an experimental real-time update fashion with both software and documentation was required. As a result, CM issues resulted (described above). These issues could have been avoided in a more formal environment, but at an increase to both project schedule and program cost. Participants should understand, discuss, and accept trade-offs. Maintain open dialog with Participants and status the Customer of both progress and issues. Collectively, we performed well here. We had a project slip, but all issues were well communicated to our Customers.

7) Understand that open systems architectures are not entirely open – Access to target architecture software and hardware was hampered (and remains hampered) due to OEM proprietary issues. The MCAP-II platform is based on a proprietary RTOS, a variant of the commercial version. The CAAS platform requires proprietary hardware. Efforts to develop and integrate common software require that the Developer have the exact environments when developing for and transferring to multiple target architectures. To address proprietary OEM hardware and software issues, a government-owned Aviation Systems Integration Facility (ASIF) is suggested. ASIF would be loaded with target architecture build suites. These build suites need to be available to all participants, both Developers and Platform Integrators, to assist with the development, integration, and validation of future common software efforts. The ASIF having the target architectures and the hardware and software configurations can communicate these configurations to all common software Developers.

Similar efforts – WDI

The Chinook Integrator is developing the Well-Defined Interface (WDI) with similar design aspects of the CDA. The WDI will first target the CH-47 Cargo platform, with the potential to be integrated and reused across the other CAAS platforms.

While the WDI has many noteworthy aspects it was not designed for reuse across the entire suite of dissimilar Aviation platforms. In anticipation, the Army's Aviation Mission Equipment may task an effort to investigate and develop the adapters for the MCAP-II platform.

Recommendations and Next Steps

PM-AME/AATD Common Software Demonstration was a two-part effort. First, it was shown that Common Reusable Software could be integrated on disparate platforms. Second, documentation was created describing the *How To and Lessons Learned* [4,5] both from the Developer and Integrator viewpoints. Therefore, AME (and/or other DoD organization) can leverage the effort if they decide to move forward with additional Common Software efforts.

With these lessons learned and the success of the common software demonstration, next steps could include those to improve and expand on the CDA concept and approach. They could move the CDA-RC demonstration forward from lab experiment to an airworthy Reusable Software Component (RSC) for AME Communications.

Along these lines, the FAA's AC 20-148[1] identifies guidelines for RSC. The guidelines call for a Reuse Plan that includes a Project Plan. The project plan must overtly claim the intention of complying with AC 20-148 and Army Aviation certification requirements. This requires the compliance with the RTCA DO-178B [7]: planning, documentation, standards, testing, tracing, configuration management, auditing, etc. Naturally, the safety aspects of the RSC use on require significant planning. Within the list above, issues we observed with respect to *Make the Documentation Useful for Integration* (4), *Address the Programmatic Issues Early* (5), and *The Developer-Integrator Process needs to Mature* (6) would be addressed.

The Reuse Plan would also identify the types of systems and their intended scope of use – addressing the *Understand the*

Target Platform (1). The RSC project plan must identify the systems it is to be used on and speak to the implications this scope of use has on the RSC design. Specific direction must be given to each system (various aviation platforms) incorporating the RSC or, at a minimum, the issues that are likely to occur and the various issues affecting each platform identified. This latter part would address *The Open Systems Architectures are not entirely Open (7).*

Processes and Visions for Reusable Software Components

The Army's Assistant Secretary of the Army (Acquisition, Logistics, and Technology) is spearheading efforts [8] for "rapid equipping," "rapid fielding," and transforming the Army's acquisition processes. PM-AME's efforts with CSI, CDA-RC, and WDI are moving toward those goals.

The common software demonstration could serve as the cornerstone for AME's transformation toward its Common Software Initiative and goals for common reusable aviation software across its Aviation Fleet.

As mentioned, the planned goal for CDA is 100% reuse, such that one piece of software may be developed, tested, and certified then reused across multiple disparate platforms as described in the FAA circular AC 20-148 – Reusable Software Components [1].

A vision was presented during the CWG Demonstration. It was a vision of Reusable Software Components (RSC), a process for how common reusable software could be produced, tested, and integrated across the Aviation fleet in a cost-effective manner.

The process, aligned with FAA's AC 20-148[1], implies third-party developers could produce airworthy reusable software components (RSC) and reusable software verification components (RVC) which meet DO-178B guidelines, build and execute system-level tests at a government-owned ASIF, then with a high level of confidence rebuild the RSC and RVC on platform-specific SILs and re-run the RVC saving both time and money. On completion, the components then proceed to flight-testing.

Through the process, an airworthy certification is achieved and an acceptance letter of the RSC and its reusable artifacts are presented back to the Developer, for reuse at subsequent platform SILs, etc.

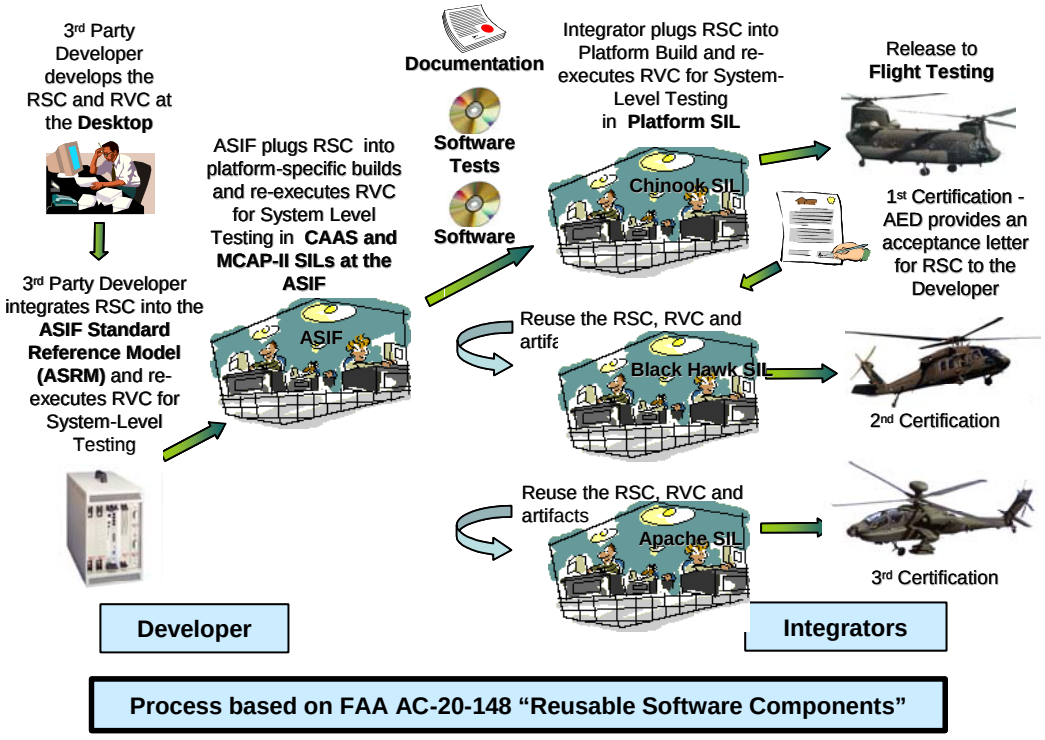


Figure 8. Vision for Reusable Software Components

Other necessary components for the vision for reusable software components are: 1) an Aviation common software repository, 2) common aviation interface standards and common middleware aviation software based on requirements that specifically call for reuse, and 3) acceptance to an airworthy certification process based on the guidelines of the AC 20-148 and that which is acceptable to airworthy DO-178B requirements. The Aviation and Missile Research, Development and Engineering Center's (AMRDEC) Software Engineering Directorate (SED) verifies life-cycle artifacts and certifies platform software. The PM-AME has been working with SED on this reuse vision.

The long-term vision for AME should include an outline of AME Best Business Practices [9, 10, 11] for not just communications, but for all of the AME Functional Areas (Communications, Mission Planning, Interoperability, and Navigation) using the CSI and CDA concepts as they evolve.

Conclusions

The Capability Driven Architecture (CDA) has been demonstrated to AME as an airworthy design for creating reusable software components. It is an open architecture for integrating and deploying new and legacy capabilities and avionics onto Army rotorcraft platforms. While architectures exist that can claim software reuse, few, if any, can claim software reuse for safety critical airworthy applications.

The common software demonstrations, which took Capability Driven Architecture for Radio Control (CDA-RC) software for two tactical radios and integrated it on three disparate Aviation platforms has been a success. Combining these two demonstrations, once piece of software operating two LRUs (ARC-201D and ARC-231) were integrated on three disparate platforms, and verified using one test suite.

The knowledge and experience gained from this demonstration has advanced the methods of common software development, and clarified a vision that will further the implementation of the Army's Common Software Initiative.

For additional information about CDA, contact PM-AME or Tucson Embedded Systems, Inc. at 520.575-7283x109, Mr. Dennis Kenman, TES–Army Program Manager.

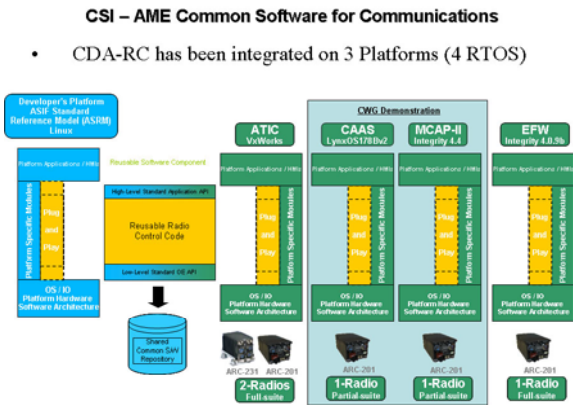


Figure 9 - CDA-RC on 3 Airworthy Platforms

References

[1] *Advisory Circular AC 20-148 – Reusable Software Components*, US Department of Transportation, Federal Aviation Administration, December 2004.

[2] *Defense Acquisition Guidebook, Chapter 2–Defense Acquisition Program Goals and Strategy*, 20 December 2004.

[3] *Army Acquisition Policy, AR 70-1*, 16 January 2006.

[4] *Commonality Working Group Common Software Demonstration, Lessons Leaned*, Tucson Embedded Systems, Inc., 20 July 2006

[5] *Supporting the Common Software Initiative*, Capability Driven Architecture – Radio Control, Reusable Software Component, Integrator's User Guide, Tucson Embedded Systems, Inc., 7 August 2006.

[6] *The ASIF Standard Reference Model (ASRM)–The Development Environment that will Enable Common Software Development for Army Aviation Aircraft*, Tucson Embedded Systems, Inc. 9 February 2007

[7] *RTCA DO-178B, "Software Considerations in Airborne Systems and Equipment Certification"*, RTCA, Inc., 1140 Connecticut Avenue, Northwest, Suite 1020, Washington, D.C., 1 December 1992

[8] "*Claude M. Bolton Jr. Assistant Secretary of the Army (Acquisition, Logistics, and Technology) Talks to Defense AT&L"*, Defense AT&L, November-December 2004.

[9,10,11] *Software Product Lines – Practices and Patterns*, *CMMI, and CMMI-AM* are all products of Carnegie Mellon Software Engineering Institute, March 2004.