The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study

Paul Clements
John Bergey

September 2005

TECHNICAL REPORT
CMU/SEI-2005-TR-019
ESC-TR-2005-019

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study

Paul Clements
John Bergey

CMU/SEI-2005-TR-019
ESC-TR-2005-019

September 2005

Product Line Practice Initiative

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work is also sponsored by the U. S. Army.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Acknowledgements

Case studies are always difficult to produce because they require inside information from people who are usually too busy, understandably, to spend time sharing with outsiders. Every successful case study is the result of extraordinary generosity on the part of the people and organizations involved. This one is no exception. This report would not have been possible without the time, energy, enthusiasm, and patience of the U.S. Army's Technical Applications Program Office (TAPO) and the Common Avionics Architecture System (CAAS) project personnel at Rockwell Collins in Cedar Rapids, Iowa. In particular, we thank Steve Overbeck, Scott White, and John Terry of Rockwell Collins and the TAPO staff.

Abstract

This report is one in a series of Carnegie Mellon® Software Engineering Institute case studies of organizations that have adopted a software product line approach for developing a family of software-intensive systems. The U.S. Army's Technical Applications Program Office (TAPO) has adopted a product line approach for the avionics software used for the Army's special operations helicopters. That software is based on Rockwell Collins' Common Avionics Architecture System (CAAS). The product line has evolved beyond its original scope and is now being adopted to include other Army aviation platforms such as cargo and utility helicopters.

This case study describes the acquisition context and organizations involved in the product line, the history behind the development and evolution of the product line, its application to the mission of the Army's special operations helicopters, the Army's motivation for adopting a product line, specifics of the product line approach, and the underlying CAAS system and software architecture. The case study also highlights the software product line accomplishments, examines the results and lessons learned from TAPO's and Rockwell Collins' perspective, and discusses future considerations.

1 Introduction

A software product line is a set of software-intensive systems sharing a common, managed set of features satisfying the specific needs of a particular market segment or mission, and that are developed from a common set of core assets in a prescribed way [Clements 02]. Organizations developing or acquiring systems using the product line approach have enjoyed significant (sometimes order-of-magnitude) improvements in time to market, cost, productivity, and quality.

This technical report is a case study of one such product line—a set of U.S. Army helicopter mission and avionics software systems based on the Common Avionics Architecture System (CAAS) architecture. Developed under the auspices of the Army's Technical Applications Program Office (TAPO), the CAAS product line represents an effort to reduce development, maintenance, and integration costs for the Army's fleet of special operations helicopters. But as this case study will describe, the scope of the CAAS product line is evolving beyond special operations and into Army aviation at large.

The two major organizations involved in the acquisition and development of the CAAS product line were TAPO and Rockwell Collins International (RCI).

TAPO is located in Fort Eustis, VA. Its mission is to use streamlined acquisition procedures to rapidly procure and integrate non-developmental item (NDI) equipment and systems for Army Special Operations Aviation (SOA), manage SOA's modifications and configuration control, provide logistics sustainment of SOA-peculiar equipment and systems, and, where applicable, transition SOA-peculiar equipment and systems to conventional Army aircraft.

The TAPO and its prime contractor, Rockwell Collins, report that their product line for Army special operations helicopters supports easy insertion of new technology (e.g., multifunction displays [MFDs] and other line replaceable units [LRUs]) and accommodates integration of subsystems by third-party developers. As a result, their product line will now be used to support a more extensive fleet of Army utility and cargo helicopters. TAPO was supported in its efforts by the Carnegie Mellon® Software Engineering Institute (SEI).

Rockwell Collins, headquartered in Cedar Rapids, Iowa, is the software integrator for the CAAS and the provider of MFDs, control display units (CDUs), and general-purpose processing units (GPPUs) shared by all three types of SOA helicopters. A Rockwell Collins Research and Development (R&D) project derived and developed the software architecture

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

for the CAAS from the Rockwell Collins Flight2 avionics architecture used in military and commercial aircraft such as the KC-135 and Boeing 767.

Rockwell Collins calls itself “a recognized leader in the development and fielding of open systems solutions, data links and self organizing network communications, advanced display solutions, and precision geo-location and navigation” [Rockwell Collins 05]. These elements play a key role in developing new technology solutions such as the CAAS.

2 History

The CAAS product line began in the late 1990s when Rockwell Collins won a U.S. Air Force contract for the KC-135 Global Air Traffic Management System (GATM). The purpose of GATM was to enable military aircraft to fly in commercial airspace by adhering to the rules that govern commercial flights. Rockwell Collins' technical approach involved leveraging its investment in Common Reusable Elements (CoRE) software and its Flight2 architecture from the company's commercial systems to accommodate an open, modular, avionics system solution incorporating common, reusable processing and hardware elements. As an outgrowth of the GATM effort, Rockwell Collins initiated an R&D project that derived and developed the software architecture for the CAAS from the Flight2 avionics architecture.

In parallel, TAPO was exploring ways to avoid the high degree of duplication in the core avionics systems of the helicopter variants (Table 1) it was supporting and that were becoming increasingly more difficult and costly to maintain.

Table 1: Supported Platform Variants in SOA Fleet of Helicopters

Platform Name	**Platform Variants**
Little Bird	AH-6, MH-6
Chinook	MH-47D, MH-47E
Blackhawk	MH-60K, MH-60L, MH60L-IDAP

Best commercial practice pointed to demonstrated significant advantages through a product line approach to software. So TAPO engaged the SEI to help investigate the feasibility of developing a product line that could support a single cockpit architecture to reduce the integration costs, development time, logistical support, training resources, and technical risk by exploiting the commonality among the systems.

Although three classes of aircraft were being supported at the time, TAPO decided to focus on achieving a common avionics system for the MH-47 and MH-60 series of aircraft and, in the future, to consider expanding the product line to include integrated systems for the A/MH-6 series of aircraft.

At that time, the two major avionics systems being used in SOA helicopters were the Cockpit Management System (CMS) for the MH-47 D and MH-60 L and the Integrated Avionics System (IAS) (a product of another contractor) for the MH-47 E and MH-60 K.

To understand the features that the product line should encompass, a team led by the SEI was formed to examine the existing products or near products in order to establish the scope of the product line. To perform the scoping, the team performed a mission investigation, product investigation, and context analysis for the different avionics system variants identified in Table 1. A scoping report was produced that defined the boundaries of the product line, the external entities that will interact with the product line systems, the interactions that will occur, and how the systems in the product line will be used.

Following the scoping phase, an SEI team performed an architecture analysis of both the CMS and IAS avionics systems to evaluate the current architectures using a shortened version of the SEI's Software Architecture Analysis Method (SAAM) as well as a technical probe to learn current architecting techniques. The probe involved interviews of individuals involved primarily in the architectural aspects of the systems.[1]

The scoping report and results of the architecture comparison were used, in part, as the basis for dialoguing with potential suppliers to sort out functional differences among the systems and investigate different courses of action for creating a common avionics system using a product line approach.

Based on this dialogue and analyses of alternative courses of action, TAPO determined that the CMS system was the preferred baseline for achieving its vision for a common avionics system. CMS was based on the CAAS, which already had many of the technical underpinnings to support a product line approach.

Since the contract was awarded, other commands, agencies, and services across the U.S. Department of Defense (DoD) have learned about the CAAS product line approach and have awarded contracts to Rockwell Collins to integrate a CAAS-based avionics system in their helicopter platforms. LTC Brian Thompson, the director of systems integration and maintenance for the 160th Special Operations Aviation Regiment (SOAR), and David Ward, a test and evaluation officer for the 160th SOAR, described it like this:

> "What began as a developmental program for SOA has grown into a cockpit modernization program that crosses military service lines. At the direction of the Army Chief of Staff, both the "big Army" utility and cargo helicopter programs will integrate the CAAS cockpit into their production lines. The Utility Program Manager will field CAAS with the UH-60M in the 2010 timeframe, while the Cargo Program Manager will field CAAS with the CH-47F in 2007. The United States Coast Guard has begun development of a CAAS cockpit for their HH-60. Both the Presidential Helicopter Squadron and the Navy H-53 Program Office have looked at the work that is being done with CAAS. Ultimately, aircraft commonality could extend across the Department of Defense" [Thompson 05].

[1] The probe was a precursor to the SEI Product Line Technical Probe℠ (a service mark of Carnegie Mellon University).

The 160th SOAR currently maintains a fleet of 10 separate helicopter series. Its modernization goal is to cull that fleet down to three primary airframes: MH-47G, MH-60M, and A/MH-6M. The vision is that the CAAS will be common to these three airframes, thereby reducing the logistics, maintenance, and training burden on the Regiment.

An article in the February 2003 issue of *National Defense* magazine described the upgrade program as follows:

> "During the next several years, U.S. Army Special Operations helicopters will be upgraded with modern avionics, designed to improve the capabilities of the aircraft, while making them easier to maintain, officials said.
>
> The technology is called common avionics architecture system—or CAAS—and will be installed on Special Operations MH-47G Chinooks, MH-60M Black Hawks, and MH/AH-6M Little Birds cockpits.[2] If the program goes as planned, more than 140 helicopters could be upgraded by 2010.
>
> U.S. Special Operations Command officials said that common hardware and software in all three aircraft should reduce the logistics demands on aviation units and simplify training. The open avionics architecture also will make it easier to upgrade the aircraft with software from third-party vendors.
>
> Most important to Special Operations aircrews, the CAAS will help manage cockpit workload and enhance situational awareness.
>
> Rockwell Collins received a contract in April—worth up to $40 million—for the avionics upgrade. Plans call for CAAS flight tests to begin in the summer of 2003 and first production deliveries are scheduled for fiscal year 2004…
>
> The 160th Special Operations Aviation Regiment…currently operates MH-47Es, MH-47Ds, MH-60Ks, MH-60Ls, and AH/MH-6s. The mixed fleet uses two proprietary avionics systems, both integrated on MIL STD 1553B databuses. They are costly to maintain, as a result of obsolescent hardware and monolithic code software.
>
> The Lockheed Martin integrated avionics system of the MH-47E and MH-60K is programmed in Jovial. The Rockwell Collins cockpit management system in the MH-47D, MH-60L, and MH-6/AH-6 uses ADA 83. Fleet upgrades must be written in both languages, and the proprietary architectures require costly regression testing when functions are changed or capabilities are added…

[2] Since this article was written, the Army has changed its plans; currently, CAAS is not planned for installation on Little Bird aircraft.

The CAAS can be described as a flexible software architecture built on lessons from the personal computer industry. Multiple isolated applications can run on a single processor, and the open system architecture with a Windows-based operating system simplifies connectivity and support. CAAS software is written in ADA 95 and partitioned into logical modules that can be modified with less expensive regression testing. The Army retains the rights to seek competitive upgrades from third-party vendors, and changes would be made with a standard software toolkit…" [Colucci 03].

3 About the Application

The CAAS software architecture is being used to support the creation of a software product line for the special operations MH-60M Black Hawk (Figure 1) and MH-47G Chinook (Figure 2). In the future, that product line may potentially include the MH/AH-6M Little Bird (Figure 3) cockpit avionics software. Each system (i.e., each member of the product line) acts as a central information system and controller for those systems within the aircraft that are information based. That includes systems that

- control the interaction between the pilot/crew and aircraft systems, flight controls, and communication and information services such as digital modems, sensors, and navigation systems
- handle the customization and setting of system parameters such as aircraft performance, pilot preferences, and weapons loads
- handle flight management and guidance
- handle elements of the engine, torque, and transmission

Figure 1: U.S. Army Blackhawk Helicopter (U.S. Army Photo)

The Army Special Operations Command has established the following missions in which SOA aircraft play a critical role—missions, therefore, that CAAS systems must support:

- counter weapons of mass destruction (WMD)
- locate/identify/recover/transport/turnover WMD
- conduct anti-ship missions
- conduct area reconnaissance
  - assessment

- environment

- conduct raids
- disable/destroy targets (direct action)

Figure 2: U.S. Army CH-47 Chinook Helicopter (U.S. Army Photo)

For the most part, these missions involve the infiltration/exfiltration of troops in support of an overall military objective. Another way of looking at missions is to consider the motivation behind creating the Integrated Avionics Cockpit (IAC) product line. From this perspective, SOA software must address several concerns:

- extended range – It must support better aircraft performance and fuel usage.
- reduced workload – It must perform functions normally carried out by the pilot or crew.
- enhanced flexibility – It must support easy integration of new electronics systems.
- self-protection – It must provide countermeasures against infrared (IR) or radio frequency (RF) weapons.
- enhanced communications interoperability/connectivity – It must provide new and better interfaces to radio systems, both analog and digital.

- improved situational awareness – It must integrate and fuse sensor systems to provide a complete picture of the combat situation.
- improved navigation accuracy – It must use global positioning systems (GPSs) and other navigation aids to assure time on target and target point arrival.

The AH-6 "Little Bird" is a light attack helicopter (Figure 3) that is flown by units of the 160th SOAR for close air support and direct attack in adverse weather and at night. The current AH-6 has some of the same services as the MH-47s and 60s, but, unlike them, it's integrated as a cockpit system. Future plans may potentially include upgrading the AH-6, along the lines of the MH-60L but with fewer capabilities.

Figure 3: U.S. Army AH-6 Little Bird Helicopter (U.S. Army Photo)

Many of the system integration capabilities of the MH-47E and -60K—such as those for fuel and some facets of flight management—are not present in the MH-47D and -60L.

The major services include

- mission management services
- communications services
- aircraft management (what the pilot needs to know to fly)—instrument displays; fuel; power train displays; warnings; cautions and advisories; temperature
- navigation services (position)
- avionics management—turning items on and off/initializing, implemented coupled/commanded flight, getting status
- sensor control—how to interact with a sensor (including threat location)
- guidance and flight director
- survivability
- weapons management
- data transfer (threats, flight plans, navigational aids, and charts)

The context diagram shown in Figure 4 shows the system in terms of a set of interacting services. When the user (a maintainer or member of the air crew) requests particular services from the system, one of three things can occur:

1. Those services are satisfied totally by the system.
2. Those services are satisfied partially by the system in connection with external entities (e.g., external sensors providing information displayed on a digital map).
3. The system acts merely as a conduit through which external services are accessed.

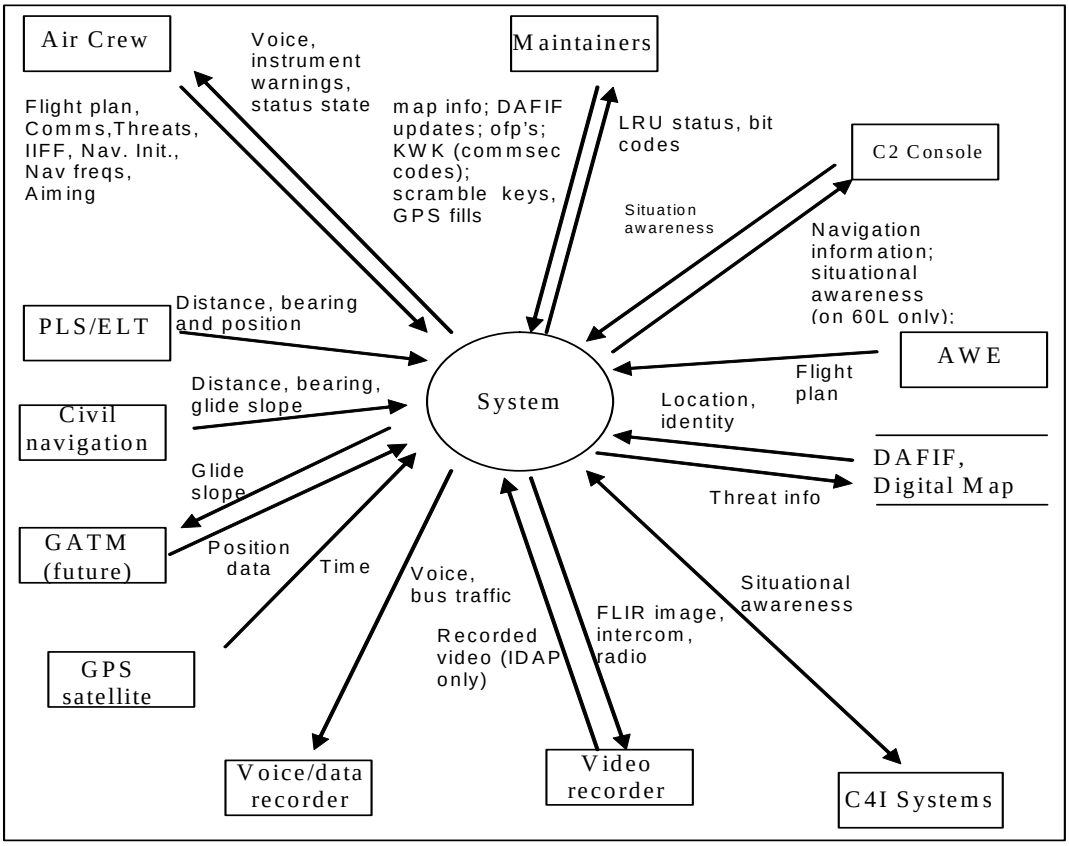

Figure 4: System Context

The system plays an integration and control role inside the aircraft. It not only provides the crew with the usual access to the aircraft's systems, but due to its privileged position, the system is able to offer services that otherwise would not be available due to the lack of integration between the subsystems involved. For example, in coupled mode, the system is capable of flying the aircraft, while in commanded mode it offers directions to assist pilot operations. Also, the system will contain pilot settings that may be used in radio adjustments or engine performance/optimization.

The system simplifies access for almost all the aircraft's systems and provides to the crew, in the friendliest way possible, external information and services. The services that are provided are quite extensive and include mission management, communications, aircraft management, navigation, avionics management, and weapons management.

Even though the system is a central element in the aircraft, it must still support alternatives and options. The main functionality of the aircraft is the movement and safety of customers—a mission that must take precedence over system capabilities. In other words, all the services and functionality offered by the system should be optional to the crew. Even in the case of a complete failure of the system, the crew must still be able to access the base functions of the aircraft (e.g., determine aircraft status, fly the aircraft, and operate communications systems). The system must also be configurable and accessible by maintenance personnel.

An article in the February 2003 issue of *National Defense* magazine says the following about the 160th SOAR's aircraft:

> "The aircraft of the Army's 160th Special Operations Aviation Regiment (SOAR) are uniquely equipped to fly long missions in hostile airspace at night and in adverse weather. In one typical mission early in Operation Enduring Freedom last year, an MH-47E carried a Special Forces A-team from a forward support base in Uzbekistan to a rendezvous with anti-Taliban forces in Afghanistan. The Chinook crew used multi-mode radar to find a way through mountains in zero visibility, refueled at low altitude from an Air Force MC-130P tanker and exercised aircraft survivability equipment to counter an apparent air defense threat. The MH-47E then penetrated a dust storm before climbing over mountains into solid cloud.
>
> From 16,500 feet, the Chinook crew again used terrain following radar to let down into the target area and deliver the special operators. On the return, they descended from altitudes near 20,000 feet in zero visibility, refueled once more from an MC-130, and again used radar to negotiate the mountains. Overall, the mission lasted 8.3 hours, including 6.3 hours in adverse weather over hostile territory.
>
> Long, high-workload missions are commonplace for the highly trained aviators of the 160th SOAR. However, the fleet is suffering from age and attrition. The MH-47E with its extra fuel and multi-mode radar is the most requested platform in the Special Operations community, but two of the helicopters were lost and another severely damaged in 2001. Despite incremental improvements, all the helicopters of the 160th are due for recapitalization…
>
> Among the features that help reduce the crew workload is a "precision flight director" with an altitude-hold function for long transits. An automatic approach to the hover function sets the descent rates in desert brownouts or other adverse conditions. A ground speed-select function helps synchronize the arrival of the aircraft with other events to better coordinate assaults.
>
> The CAAS displays help improve the crews' situational awareness by providing data from both on-board and external sensors, Rockwell officials

said. The CAAS general-purpose processor can merge two video inputs into one display. Imagery from an unmanned drone may be injected into a map display to help identify targets. Through the improved data modem, near real-time intelligence data downloaded from orbiting JSTARS (Joint Surveillance and Target Attack Radar System) or other sources should help pilots avoid threats or weather and adapt to changing battlefield situations. While the cockpit still has two control/display units, pilots will be able to change flight plans and radio presets through the multifunction displays, without going head-down to the control/display units" [Colucci 03].

The Operational Flight Program (OFP) comprises some 1.1 million source lines of Ada code (SLOC). Of that, the Flight Management system is about 120K SLOC, communications systems account for some 250K SLOC, and Tactical Situation Awareness systems consume some 75-80K SLOC.

4 Army Motivations for the Product Line Approach

The vision for the CAAS was to create a scalable system that would meet the need of multiple helicopter cockpits to address obsolescence and modernization issues and use a single, open, common avionics architecture system for all platforms to reduce the total cost of ownership (TCO). The driving tenets used in the development of the CAAS approach were open system architecture, variability isolation, connectivity, supportability, and modularity—all of which are essential to providing an economical TCO.

The product line approach was motivated by the following problems the Army was experiencing with its special operations forces helicopters:

- inability of avionics to accommodate growth requirements
- high nonrecurring engineering (NRE) costs to upgrade or add new functionality
- lack of preplanned avionics upgrades to support new functionality and/or component obsolescence
- unacceptable levels of aircraft availability
- high installation costs
- limited opportunity for third-party development
- diminishing manufacturing sources
- application of commercial off-the-shelf (COTS) software without adequate consideration of the military environment

Simply put, the Army wished to avoid paying multiple development and integration costs each time a new device, enhancement, or update was incorporated into the helicopter fleet. The Army reasoned that a single architecture would eliminate this multi-integration penalty.

Overall, the goal of the CAAS is to create a scalable system that meets the need of multiple helicopter cockpits to address modernization issues. The concept pursued is to use a single, open, common avionics architecture system for all platforms to reduce the TCO. The approach is based on Rockwell Collins' CMS system in its Flight2 family of avionics systems, augmented with IAS functionality.

The four primary business drivers for the CAAS are

1. Minimize systems acquisition and sustainment costs.
2. Reduce the TCO over the life of the avionics system.
3. Modernize with minimal impact to Army Special Ops readiness.

4. Ensure that the new CAAS systems work well with other programs.

Strategies to achieve these drivers include

- Leverage existing (IAS/CMS) code and documentation where practical.
- Aim for a “plug and play” architecture where hardware is portable between platforms.
- Move away from single proprietary components and use standards where appropriate.
- Provide cross-platform commonality (hardware, software, and training).
- Reduce the logistics base.
- Use the Service Life Extension Program (SLEP) as a fleet modification/installation center as much as possible.
- Leverage other platform developments (both commercial and DoD).
- Employ a system infrastructure or framework that facilitates system maintenance and enhancements (including third-party participation in these activities).

Other business drivers that play a role include the constraints on the system and its architecture. In the CAAS, these constraints include a set of applicable standards, including

- Aeronautical Radio, Inc. (ARINC) 664, 661, and 429
- DO-178B
- MIL-STD-1553B
- RS-232 and RS-422
- DoD Information Technology Standards Registry (DISR)
- Common Object Broker Request Architecture (CORBA)
- POSIX, Institute of Electrical and Electronics Engineers (IEEE) STD 1003.1
- OpenGL

Modifiability (as is often the case, particularly in product line systems) plays an important role in the CAAS, manifesting itself in several ways. With regard to the CAAS, modifiability refers to the following capabilities:

- growth
- testability
- openness
- portability of hardware and software across different aircraft platforms
- reconfigurability

- repeatability—Every system must look like every other system and produce the same output on every platform.
- supportability—A third party must be able to maintain the system.
- reuse
- affordability

5 Product Line Approach

To meet these challenges, the vision for the CAAS was twofold:

1. to address obsolescence and modernization issues by creating a scalable system that would meet the needs of multiple helicopter cockpits
2. to reduce the TCO by using a single, open, common avionics architecture system for all platforms

Figure 5 illustrates this vision. An individual helicopter system is built, in classic product line fashion, by using the product line architecture and other core assets (including software components) that are maintained in a core asset library. The system is built using a production plan that explains how to turn the core assets into a helicopter product that is a member of the CAAS product line.

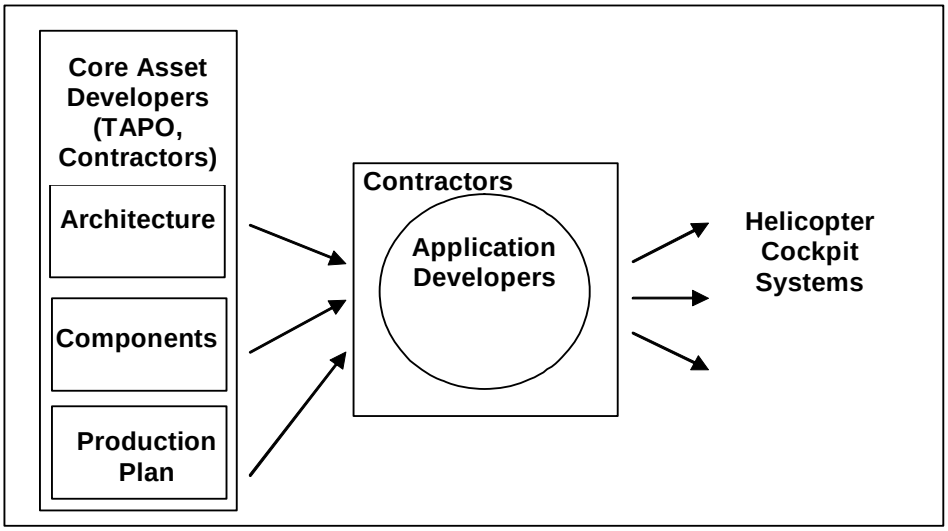

Figure 5: From Assets to Products

A product line development process means that different variations of the glass cockpit can be produced, in large part, through the use of a common collection of software and other assets. The establishment of this collection of software and other assets will be accomplished through the use of a “domain engineering” asset development process. The individual cockpit products will be created through the use of an “application engineering” product development process not discussed in this report. This product line development process is represented graphically in Figure 5.

Each product in the product line is envisioned to

- provide a set of services integrating flight management, communication, navigation, and avionics (displays, heads-up display [HUD]) capabilities
- take advantage of the cross-platform insertion of new technologies
- be capable of both efficient system integration and increased functionality with minimal impact to the current pilot vehicle interface

Rockwell Collins maintains the core asset base. However, the Army owns the Government Purpose data rights to it and so has flexibility and control over the product line and its future.

Some of the additional strengths of the software product line approach are that it supports the insertion of new technology, enables MFDs and other avionic equipment units to be swapped out from one helicopter avionics system to another with automatic reconfiguration, and accommodates integration of subsystems by third-party developers through well-defined application program interfaces (APIs). The system is compliant with the DISR requirement, using industry-standard, system-level interfaces: MIL-STD-1553 and ARINC 429 for legacy equipment, SMPTE 292 HDTV video standard for system video, and IEEE 802.3 for ethernet connectivity.

6 The CAAS Architecture

Because software architecture is a major determinant of software quality, it follows that software architecture is critical to the quality of any software-intensive system. This is especially true for a software product line; the architecture is generally regarded as the most key of all the family's core assets. This section describes the hardware and software architecture for the CAAS family of systems.

The CAAS embodies the following architectural precepts:

- an Open System Architecture (OSA) using published and controlled interface definitions, such that its hardware and software components can be replaced or upgraded with alternate components
- variability isolation to accommodate changes in the system over its life cycle, such that the impact of change is isolated to the smallest system component
- use of layers and partitions with widely accepted interfaces to isolate system components
- redundant software using master/slave protocol where every application is resident on every box and some applications are active on multiple boxes to support quality attributes such as availability and safety
- use of application templates across applications, common software, and CoRE to support reusability, modifiability, repeatability, and affordability
- use of commercial standards including ARINC 661 (cockpit display system interface standards), POSIX, CORBA, IEEE P1386/P1386.1 (common mezzanine card families draft standards), OpenGL (graphical interfaces standards), and DO 178B (software considerations for airborne systems) to enhance portability, maintainability, and modifiability
- LynxOS-178 based on the flight-ready POSIX operating system (with standard POSIX API and Ada 95 support) to encapsulate and manage any interaction with the computing platform and provide DO-178B, Level A design assurance

Other features being supported include a high-integrity, ethernet local area network (LAN) with COTS general-purpose processors, video processors, and graphics engines. The architectural impact on the total life-cycle cost is that it will reduce the NRE costs of upgrading or adding new functionality and allow the use of all available processing reserves when upgrading the system without drastically rerouting system data. A software application developer's toolkit will be available to third parties, so they can integrate their software applications. This toolkit includes

- references to COTS APIs and COTS toolkits
- a set of Rockwell Collins APIs for interfacing to applications developed by that company

- a set of system integration tools

Overall, the CAAS architecture is distributed, with each computational platform having the same hardware/software infrastructure but different individual applications. LynxOS-178 is the basis for the infrastructure, and it satisfies both the ARINC 653[3] and POSIX standards; LynxOS-178 provides "brick wall" partitioning of memory and processing. The applications reside in LynxOS-178 partitions, which are created at load time and cannot access memory in the other partitions. Each partition receives a strict cycle of processor time: unused time within a partition's cycle is spare and remains unused. If a processor overruns its time, a fault monitor detects the occurrence and can either cause the partition to fail or allow it to restart where it left off on the next cycle. The applications can execute as a number of POSIX threads within a partition. All communications between partitions takes place through calls to LynxOS-178 and the Ethernet network. The goal is to have a "plug and play" architecture with respect to applications and processors.

The architecture is highly redundant, containing both communication line and processor redundancy, with well-defined fault detection schemes and failover philosophy. However, not all processors are redundant, so some functional degradation can occur with a single failure but not to the critical functions. A "hot standby" software redundancy scheme is used.

6.1 The CAAS System Architecture

The scaleable CAAS uses one Power PC 750 processor in each CDU and two Power PC 750 processors in each MFD. The open system architecture is meant to simplify connectivity and support including the ability to swap LRUs between platforms in the field. The use of this common avionics hardware and software across the aircraft will reduce the logistics demands on aviation units and simplify training. And the commonality of software and hardware components is expected to provide the special operations forces a lower total life-cycle cost and lower costs for technology insertion and supportability.

The CAAS system architecture is depicted in Figure 6. The software of interest resides in the CDUs and MFDs. The GPPUs contain the digital map software. The Armament System Processor Panel (ASPP) contains embedded software.

[3] Standard partitioning is defined in ARINC 653. LynxOS-178 does not implement the ARINC 653 API, but it does provide 653-style partitioning support.

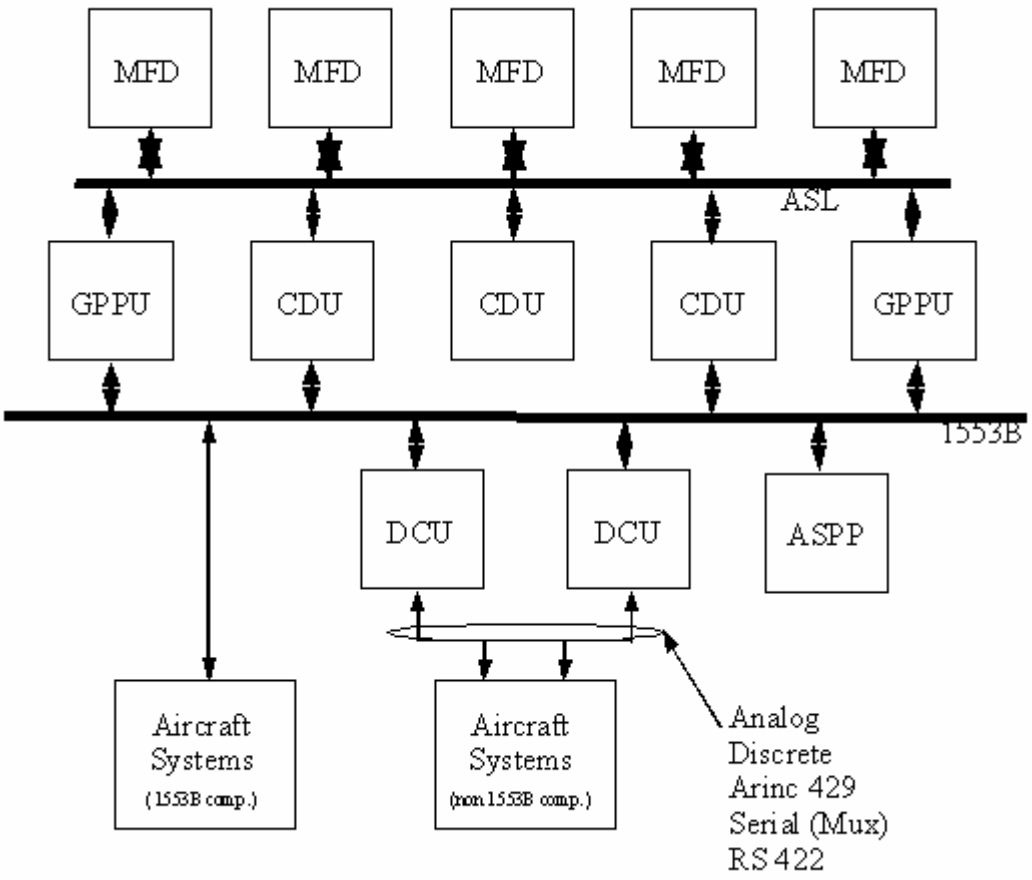

Figure 6: The CAAS System Architecture (Top Level)

The system architecture includes the following elements:

- MFDs – These are general-purpose graphical display and processing units including two processors: one display and one mission. There are four MFDs (and an optional fifth) on all aircraft except the MH-6 (which contains two). These MFDs are the primary means for displaying information to the crew. The crew's interaction with the MFD is through the bezel switches.
- CDUs – These are the primary data entry units for the system and contain a character-oriented display and keyboard. They contain a general-purpose processor used for running CAAS software. There are three CDUs on each aircraft except the MH-6 (which contains two). One CDU functions as the bus controller and one as the backup bus controller for the MIL-STD-1553B busses.
- data concentrator units (DCUs) – This is a general-purpose interface control unit used for connecting the CAAS system to devices that don't comply with the MIL-STD-1553B. Interface types include discrete, analog, ARINC 429, RS-422, and serial multiplex. There are two DCUs on each aircraft except the MH-6 (which doesn't have any).

- ASPP – This unit is responsible for low-level control of weapons and management of the hardware interfaces to those weapons. There is one on each MH-60 IDAP and MH-6 aircraft.
- GPPUs – These units are used primarily for video processing and provide digital video to the MFDs. They also contain the digital map software that generates the digital map video for display on MFDs. In addition, the GPPU contains the digital switch that serves as the full-duplex switch fabric and control for the avionics system LAN (ASL).
- ASLs – Two ARINC-664-based ethernet LANs are used in the system. They are the primary means of communications among the software elements of the CAAS. The networks are arranged as a set of ethernet segments connecting each node on the network to the full-duplex switch fabric. Each network segment is restricted to one user node and the switch connection, to eliminate collisions on the network. The switch receives message traffic sent by each user node and routes it to the segments that contain the nodes that are to receive the message. The network topology is a star.
- MIL-STD-1553B serial data bus - Two dual redundant 1553B busses are used to handle communications between the CAAS software and the 1553B-compliant devices mounted on the aircraft. One CDU acts as the bus controller, while a second acts as the backup bus controller for the networks.

6.2 Software Architecture

The primary structural mechanism used by CAAS software is layering (shown in Figure 7). Mission-related functionality lies, for the most part, in an application layer that is the top layer in a software stack. The applications make use of system services through an API. All access to lower level system capabilities is routed through this API. System services communicate to hardware through a device driver layer. Hardware characteristics are abstracted for all devices by the device driver layer.

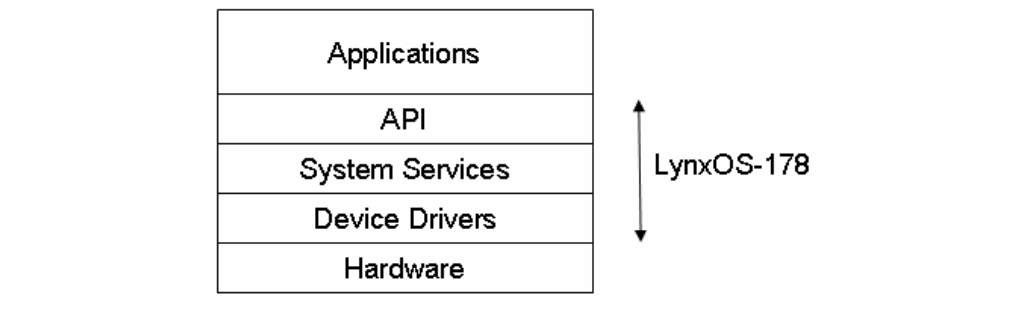

Figure 7: CAAS Software Layering

The API and System Services layers shown in the diagram above are provided by the LynxOS-178, which sits on top of the device drives for things like the 1553 bus and ARINC 429 Input/Output.

6.2.1 LynxOS-178 Kernel

The LynxOS-178 kernel is the operating system kernel for the CAAS and handles resource management in the system. Each processor contains a number of memory partitions, which are scheduled in accordance with ARINC 653 using a static, time-sliced, priority-based, interruptible, cyclic scheduler.

The scheduler performs all dispatching of partitions and threads within partitions based on the schedule that is currently active. Three schedules are supported: cold start, warm start, and normal operation. The schedules are supplied from the Virtual Machine Configuration Table (VCT).

Memory management is handled mostly by the memory management unit (MMU), which is capable of detecting memory violations and generating an exception in response. The kernel services the exception, suspends the offender, and activates the Health Monitor (HM). The VCT specifies limits for memory allocation to each partition. These limits are enforced by the MMU and kernel, preventing a partition from accessing memory not allocated to it. Blocks of memory are allocated to partitions at initialization time, and no further changes in memory allocation occur.

6.2.2 Applications

The primary component identified for applications is the partition. A partition is a term tracing to the operating system kernel, which recognizes a partition as the primary unit of resource allocation and management. A partition is a container for a collection of lower level components including both mission-related software components and infrastructure components. Thirteen partitions are distributed among the MFDs and CDUs:

1. CDU display manager
2. MFD display manager
3. CDU input/output (I/O) manager
4. flight director
5. flight manager
6. system manager
7. mission sensor manager
8. communications manager
9. weapons manager
10. EICAS manager
11. tactical situation awareness
12. CDU MP VM0 (HM for CDU)
13. MFD MP VM0 (HM for MFD)

Most partitions are structured according to a common model discussed below. The HM is a partition that lies at the same level as application partitions but is effectively part of the software infrastructure.

6.2.3 Partition Model

The internal structure of partitions, which should conform to the model shown in Figure 8, provides an environment in which the application functionality lives.

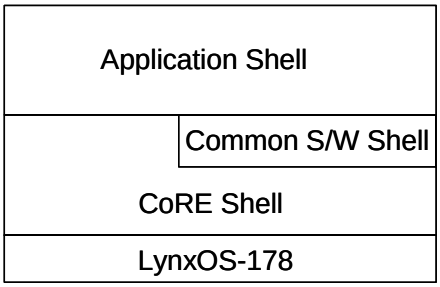

Figure 8: Computer Software Configuration Item (CSCI) Model, Layered View

The application shell provides a number of component types and component implementations for use by the applications (see Figure 9). Some of these component types are extended or completed by developers of the partition—that is, the component type is intended as an abstraction that is instantiated for use. Some components are used as is; their implementation is provided as part of the shell.

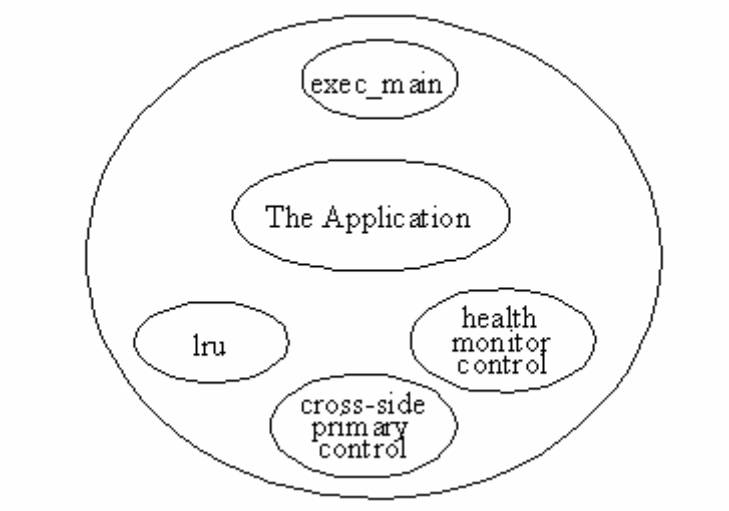

Figure 9: Application Shell Components

Four of the components (i.e., exec_main, lru, health monitor, and cross-side primary control) provide essential services to the application, which performs the appropriate avionics functionality.

The CoRE Shell is a collection of infrastructure components that supports the application Computer Software Configuration Item (CSCI). This shell is a set of common reusable components developed by the contractor for use on multiple programs. It contains some basic infrastructure services needed by avionics systems.

6.2.4 Coordination Mechanisms

The coordination mechanisms defined for the CAAS are relatively limited in number. They are concerned mostly with the management of system resources and communication between software elements.

The scheduling of processes is performed by the LynxOS-178 kernel in conjunction with the exec_main component of the partition shell. It is a preemptive, priority-based, cyclic scheduler with a fixed schedule. The kernel scheduler is responsible for dispatching all processes and threads within processes. It receives a hardware signal at a periodic rate with a period of 1 millisecond and uses that signal as a trigger to evaluate the schedule.

Intrapartition communications is restricted mostly to procedure calls. Data transfers are accomplished via parameters provided with the calls. Although it's discouraged, Ada rendezvous is used in some CSCIs. There are no standards or conventions of architectural significance associated with intrapartition communication. Interpartition communication uses two mechanisms:

1. The primary connection-oriented service is the remote service provider/remote service interface (RSP/RSI).
2. The Trivial File Transfer Protocol (TFTP) is a simplified, low-quality-of-service method for transferring blocks of data between two software elements. In the CAAS, it is intended primarily for transferring configuration files or mission data files between the bulk loading components (DR-200) and the software that uses the transferred data.

Memory management is based on the virtual machine concept where memory is allocated and managed on a partition basis. Most management of memory access is performed in hardware. Memory address violations are detected by the MMU and a signal generated to the HM. The offending partition is suspended on a violation. Several types of memory are managed including semipersistent and persistent. Allocation is made during initialization, and memory is never returned or further allocated during runtime. Further management of the memory blocks allocated to a partition is the responsibility of the partition. Data coupling between partitions using shared memory is not used.

Display management is responsible for getting the information rendered on the glass.

6.2.5 Reliability

The CAAS software architecture supports reliability with a variety of mechanisms. The most important architectural mechanism is redundancy. The current implementation of the CAAS uses dual redundant (primary and hot spare) partitions, although the implementation of components supports a higher level of redundancy. The redundancy discussed here applies to partitions other than display managers. Display redundancy works differently and is discussed in more detail below.

Reliability is achieved through the following architectural approaches:

- software redundancy: The strategy for software is to use a primary unit (master unit) and a hot backup (slave). Partitions use a watchdog service provided by the HM. A failure of a primary to stroke the watchdog within a specified time period can trigger a switch of the backup to the primary role.
- health monitoring: The HM provides a number of capabilities to enhance system reliability. It monitors memory usage and responds to any memory protection violations by killing the offending partition. In addition to monitoring software versions to insure that compatible versions are loaded and checking for software corruption, HM performs background-initiated built-in test of processing elements.
- memory persistence and warm start: Restarts due to momentary power interruptions are supported by semipersistent memory storage for state data. When the system detects a power interruption of less than two seconds in duration, it directs a warm start using state data stored in semipersistent memory.
- display redundancy: A measure of reliability for the display of information is afforded by the transparency of display pages to the physical medium on which it is displayed. Each CDU has a separate copy of the CDU display manager, and each MFD has a separate

copy of the MFD display manager. Any page generated by a partition can be displayed on any appropriate display device (CDU or MFD) so that the loss of a particular hardware device does not preclude the display of any page in the system. Users command which pages are displayed and where. The redundant displays do not assume a master/slave relationship but simply provide multiple devices capable of displaying (or inputting) the same information.

- virtual circuit service: Communications reliability is provided by allowing connections to be established using a virtual circuit service. This service (provided by the Transmission Control Protocol [TCP]) provides for guaranteed in-order delivery of traffic. It keeps track of messages received and retransmits lost messages. It also monitors and reports the status of already established connections.

6.2.6 Architectural Approaches

Because architectural approaches exhibit known affects on quality attributes (for instance, redundancy increases availability, whereas layering increases portability), explicitly identifying the approaches provides deeper understanding of the architecture.

The overarching strategies used in the CAAS are strong partitioning of applications and structuring the software as a series of layers. The POSIX operating system provides the primary partitioning mechanism, establishing inviolable timing and memory walls between applications to prevent conflicts. For example, an important result of this design is that an application that for some reason overruns its processing time limit cannot cause another application to be late. An application that overruns its memory allotment cannot impinge on another application's memory area.

Applications, hence, represent encapsulations. One of the strongest aspects of the CAAS architecture is that this design allows applications to be changed more or less independently from each other. The fact that applications do not depend on their hardware location makes adding new applications a straightforward process.

The overall list of CAAS software architectural approaches is given below, along with the quality attributes each one nominally affects:

- consistent partitioning strategy: definition of a partition, "brick wall" partitioning (availability, safety, modifiability, testability, maintainability)
- encapsulation: Encapsulation is used to isolate partitions. Between partitions, applications can only share state via the network. RSI and RSP are also examples of encapsulation that isolates the network implementation details. (modifiability, availability)
- interface strategy: The strategy of accessing components only via their interfaces is strictly followed. Besides controlling interactions and eliminating backdoor exploitation of changeable implementation details, this strategy reduces the number of inputs and outputs per partition. (modifiability, maintainability)

- layers: Layers are used to partition and isolate high-level graphics services. (portability, modifiability)
- distributed processing: A client server approach is used to decouple parts of the system providing separate functionality. Decoupling is also achieved through the use of information broadcast. (maintainability, modifiability)
  - Both User Datagram Protocol (UDP) and TCP are used, and rules are established for each. With UDP, data consumers issue a warning when data does not arrive.
  - Access to sockets, bandwidth, and data is guaranteed.
- virtual machine: A flight-ready POSIX-consistent operating system with a standard POSIX API and Ada 95 support providing Level A design assurance (modifiability, availability)
- HMs: HMs check the health of CDUs and MFDs. (availability)
- use of commercial standards: Standards include ARINC 661, POSIX, CORBA, IEEE P1386/P1386.1, OpenGL, and DO 178B. (portability, maintainability, modifiability)
- locational transparency: Applications do not know where other applications reside, and, hence, are unaffected when applications migrate to other hardware for schedulability or load-balancing reasons. Location is bound at configuration time. (portability, modifiability)
- isolation of system services: This is a by-product of the layering strategy. (portability, modifiability)
- redundant software: For flight-critical functions, redundant software is introduced using a master/slave protocol to manage failover. (portability, availability)
  - Every application is resident on every box. (portability)
  - Some applications are active on multiple boxes. (availability)
- memory and performance analysis: Partitions are cyclic. However, Rate Monotonic Analysis (RMA) is used to assign priorities to threads within partitions. The result is assured schedulability. (performance)
- application templates (the shell): A standard template for applications incorporates applications, common software, and CoRE and ensures that complicated protocols, such as failover, are handled consistently across all applications. (reuse, modifiability, repeatability, affordability)

7 Results and Lessons Learned

7.1 Expanding the Product Line Beyond SOA

As is often the case in software product line success stories, developing organizations are able to capitalize on the ability to quickly and agilely produce systems within the original scope (here, Army SOA) and bring that capability to bear on "nearby" application areas.[4] Here, both of the organizations involved (the U.S. Army and Rockwell Collins) were able to parlay the CAAS product line for SOA into new opportunities.

As indicated previously, the original scope of the CAAS was limited to the MH-47s and MH-60s of Army SOA. Once the benefits of the product line approach became clear and TAPO established that such a path was possible and feasible, the program began to attract attention not only within the Army but also across the DoD community.

As of the writing of this report, the CAAS has been chosen for the U.S. Coast Guard's HH-60, the U.S. Navy's VH-60, and the Army's CH-47 Chinook, new Armed Reconnaissance helicopter, and UH-60 Blackhawk helicopters. A slight variation of the CAAS architecture has also been selected for the Navy's CH-53 Sea Stallion helicopters. Table 2 shows the approximate number of aircraft involved.

[4] Examples of such organizations include the following: Cummins, Inc. took its product line of automotive diesel engines and successfully entered the industrial diesel engine market. Motorola turned its product line of one-way pagers into a product line of two-way pagers. CelsiusTech used its product line of shipboard command and control systems to enter the air defense system market. Another division of Raytheon took its product line of government satellite ground control stations and launched a product line of commercial satellite ground control stations.

Table 2: Potential Number of Aircraft in TAPO's Product Line

Service and Platform	Projected Number of Aircraft
U.S. Army SOA (various platforms)	100+
U.S. Army CH-47 Chinook	300-400
U.S. Army UH-60 Blackhawk	1300
U.S. Navy VH-60	10-15
U.S. Coast Guard HH-60	42
Armed Reconnaissance Helicopter	320
TOTAL	**2072-2177+**

The product line scope, while bounded by the aviation types it will support, is now changing. More variability is being introduced by technology insertion plans to accommodate an increase in avionics platforms that will draw from the same logistics pool of hardware and software components, from an introduction of new system features traditionally used by service-oriented architectures, and from the Army's desire to strategically reuse software components across dissimilar platforms. One driver for this variability is that the Army expects to upgrade its CH-47D helicopters to CH-47Fs, and some of them will be converted into MH-47G.s. The commonality of software and hardware components is expected to lower the total life-cycle cost and lower costs for technology insertion and supportability.

A Software Application Developer's Toolkit is available to third parties to help them migrate and integrate their software applications on CAAS processing platforms or incorporate third-party equipment using dissimilar processing platforms. The toolkit also provides the Army with the potential to competitively acquire software from other suppliers.

7.2 Results

Because of the project's timeline, it is too early yet to claim definitive cost savings. But both TAPO and Rockwell Collins agree that the following savings are forthcoming:

- **software maintenance contracts:** Currently, TAPO supports two maintenance contracts worth about $3-4 million annually. Once the CAAS is deployed across the entire SOA fleet, one of them will be eliminated.
- **development of new functionality:** To implement new functionality, the software had to be developed twice in different languages corresponding to the two different target platforms. All platforms will now benefit from a one-time development cost for implementing new functionality. Paying for development only once will result in significant savings because it costs $3-5 million to develop, test, and release each major functional update.
- **integration costs:** Each new device or piece of functionality has to be integrated and tested twice. Once the CAAS is deployed across the SOA fleet, each enhancement will only have to be integrated once. And integration should be easier because the product line

is using open standards (e.g., ethernet). The cost of integrating new functionality is $3-5 million, and TAPO supports a couple of these integrations each year. That cost should be halved due to the implementation of the CAAS.

- **documentation costs:** Documentation costs currently run $2-3 million per year. Using the CAAS across the family should reduce those costs substantially because separate documents describing different family members can be eliminated.
- **simulator flight training:** One simulator facility is being developed to handle both the delta and echo aircraft. Reducing the number of facilities required and being able to standardize pilot training will yield significant savings.
- **flight-test costs:** Flight testing will be reduced. Instead of flight testing both kinds of MH-60 and both kinds of MH-47, only one of each will need to be tested. Reducing the testing will save several hundred thousand dollars and require fewer aircraft to be pulled from operational service to support testing. The latter is significant because one of the biggest problems of testing is scheduling resources, pilots, and facilities.
- **training costs:** The comprehensive training program for each platform used to cost about $1 million to develop. Now, Rockwell Collins can reuse CAAS training across all platforms and then add platform-specific materials. The projected cost for training is now about $50,000 per platform.
- **reduced time to deployment:** The CAAS for the CH-47F will be fielded within 2-3 years and features over 90% commonality with other platforms in the CAAS family. This reduction represents a gain of anywhere from one to two years. The overall CAAS architecture will be fielded in five years, which is half of what previous systems took. That reduction is a result (at least partially) of the CAAS being a product line based on Flight2.
- **reduced system development costs:** A new CAAS system is priced around $10-15 million, which is about two thirds less than what a CAAS system used to cost.
- **reuse level:** CAAS systems benefit from strategic software reuse of around 80% or more.

Both TAPO and Rockwell Collins point to other results that have not yet been measured. On TAPO's side

- Critical fixes happen rapidly, due to the strong functional partitioning of the architecture.
- Airworthiness qualifications happen in a much shorter time because each qualification can be limited to the single functional partition that was changed.
- Because the U.S. government owns the data rights to the CAAS, the Army has the right to offer the maintenance and development contracts to other parties if Rockwell Collins' costs become prohibitive or the company goes out of business.
- The open architecture and adherence to industry standards makes the possibility of choosing a contractor other than Rockwell Collins feasible.
- Having a single kind of display simplifies maintenance and contracting and increases operational flexibility enormously.

Rockwell Collins, on the other hand, feels that the product line approach has made the company more competitive and allowed it to win more business. “From a competitive standpoint,” one Rockwell Collins employee told us, “it put us in a phenomenal position.” The company’s contract award record would seem to bear that out: To date, when Rockwell Collins has been the system prime contractor, it has been successful in 100% of its CAAS-based bids.

7.3 Lessons Learned: Rockwell Collins

Rockwell Collins feels that the architectural and product line approaches have worked well. Architecturally, there is an issue about the number of functional partitions in the architecture. The partitions represent configuration items. Having more items leads to a finer grained decomposition of functionality, which, in turn, leads to higher levels of reuse and systems that are easier to test and get through airworthiness qualifications. However, having more items also means there are more configuration items to keep track of and maintain, more complexity, and a steeper integration curve. There is also an upper limit to the number of partitions, based on the virtual memory space and sockets available. From a system standpoint, the partitioning of the software allows Rockwell Collins to quickly effect fixes and certify the software. The CAAS currently features about a dozen partitions; the architects feel that 15-20 might be a better number.

Rockwell Collins feels that it was not aggressive enough in demanding stable, detailed requirements. Were the company to do it over again, it would “demand requirements, not take ‘no’ for an answer, and echo-check them back to the customer” for validation. In addition, all parties would understand that a change in requirements would lead to a change in schedule. Their desire to handle requirements this way is independent of a product line approach.

Rockwell Collins feels that more prototyping would have been helpful. The customer, one Rockwell Collins employee said, always prefers to see (not read about) what they are going to be acquiring. The company’s prototyping capability is increased now, and future customers should enjoy that capability more. The Coast Guard is, in fact, concretely identifying its needs by building its own prototypes and showing them to Rockwell Collins. One day, customers might even be able to use actual CAAS source code as part of their prototypes.

On the acquisition side, Rockwell Collins would welcome any requests for proposals that required a product line approach because the company believes it is perfectly positioned to respond to them using already created boilerplate language. The company also believes that such requests would give it a competitive advantage.

Rockwell Collins would welcome the formation of a user group to jointly evolve and create requirements. The DoD has yet to create such a group. Interestingly, although many programs recognize the need and advantage of such a group, none seems able to form it. As a result, they have individually asked Rockwell Collins to develop common systems and price them

accordingly. In other words, the U.S. government has placed the responsibility of identifying and taking advantage of commonality in the hands of its contractor. Rockwell Collins is pleased to participate in such efforts because it feels it is in the company's best interest to quash any perception that it is charging full price for each version of the software.

Trade shows turn out to be excellent forums for assembling stakeholders with common requirements. In the case of the CAAS, one such forum has been the Army Aviation Association of America ("Quad-A") convention. At a recent convention, people from a Navy rotary-wing program office contacted other CAAS users and asked to meet informally. Such ad hoc user group meetings do happen from time to time, but they are opportunistic and largely unplanned. Again, the message from these groups is a plea for Rockwell Collins to help them manage their common requirements.

Although Rockwell Collins maintains the core asset library for SOA, the Navy, the Coast Guard, and the "big Army," it currently does not have a central management activity to evolve all those core assets jointly. For example, if a program needs a new version of a core asset, that program will develop that version based on the (older) common version of it. The new version will be made available to any program that deems it desirable, but the new version will not be folded back into the other programs. For one thing, other programs shouldn't be burdened with the test, integration, and airworthiness qualification costs associated with changes not specifically required by them. Hence, multiple versions of core assets are allowed to proliferate, which could eventually lead to a breakdown of the product line approach. Having multiple variations of all the core assets, maintained by separate programs, quickly comes to look like the stovepipe development that product line engineering is meant to replace. But for now, the amount of unique code among the OFPs is limited to the application partitions and totals about 10-15 K SLOC. Rockwell Collins points out that "massive individuality" could be obtained by changing up to 10% of the system, leaving 90% common across the family.

Organizationally, Rockwell Collins has a central platform software group for maintaining the CAAS core assets below the application layer. Each application partition also has a software group responsible for its development and evolution, and each of these partitions is, in effect, a mini-product line. The manager of each application can manage it as he or she sees fit. Application groups are run in a way that if new customers come and ask for a particular application in their CAAS family member, they don't have to pay for it as though it were new. So there's a business incentive to keep costs to the customer low but no central policy for maintaining commonality and variability.

7.4 Lessons Learned: TAPO

TAPO's perspective of the CAAS is somewhat different from that of Rockwell Collins. Although TAPO is fully aware that the CAAS is a product line, the organization took the point of view that SOA was acquiring *one thing*—a single software system that ran equally well on each of its rotary-wing platforms. TAPO's major technical challenges were

identifying and engineering the common requirements across platforms—displays, hardware, functions, and pilot interfaces. To support its growing user base, the Army has formed a CAAS working group for configuration control.

Programmatically, the TAPO staff members spoke about the many telephone calls they had to field from representatives of other programs who were interested in applying the CAAS to their aircraft. “Plan for that” is the lesson they offer. Ideally, they said that the U.S. government would stand up a program office across all aviation platforms that could use the CAAS, but that seems unlikely now. One staff member gave advice that was even more succinct: “Don’t be the first.” Being the “clearinghouse” for the product line has proved to be a challenging task.

To achieve the contracting flexibility TAPO sought, it secured the source code, environment, and so forth, for the CAAS. Theoretically TAPO could maintain the system itself or contract with any vendor to write an application that interfaces with the CAAS. TAPO has made sure that the CPUs and software architecture are open for just that purpose. To date, Rockwell Collins has been in the driver’s seat because of its proven process, product line experience, domain expertise, and application familiarity. But TAPO is not “married” to Rockwell Collins and can exercise its contractor freedom at will.

7.5 Looking to the Future

What does the future hold for the CAAS product line? Several avenues of interest are discussed below.

Architecturally, Rockwell Collins has successfully used an approach known as “neighbor load.” For SOA helicopters, the software applications installed are aware of all the helicopter differences. A big operational advantage is that a display from one SOA helicopter can be installed in another SOA helicopter (even one of a different type) without any required changes. That’s because the software across platforms is identical, and when it starts up, the first thing it asks is “What kind of helicopter am I in?” Then, it behaves accordingly. This approach does not scale easily across many platforms. It requires arduous negotiation of common requirements, it wastes resources, and aviation safety experts are uncomfortable flying large segments of “dead code” that could accidentally be activated with unforeseen consequences. For “big Army” applications, Rockwell Collins is proposing a scheme, neighbor load, in which a display device, once installed in a helicopter and activated, will query nearby platforms to see what software applications they’re running. Then, it will download and execute that software.

Programmatically, both TAPO and Rockwell Collins have recommended that the Army stand up a program office for a cockpit common across all helicopter applications. (A joint program office could even handle cross-service platforms.) To date, the Army has not accepted this challenge. In response, Rockwell Collins is creating a “pseudo program office”—an internal

group to merge and manage requirements as they come in. Such an office will help manage commonality and variability on a family-wide basis but may lack the needed veracity.

It would also be in the U.S. government's best interest to form and oversee a user group made up of Army and DoD platform representatives. Doing so would help coordinate the establishment of requirements and manage the needed commonality and variability across platforms. Such coordination could lead to wider acceptance of the product line and significant reductions in the TOC. In the near future, TAPO may initiate an omnibus contract for post-deployment software support to make software maintenance services available to all CAAS-based projects.

Finally, Rockwell Collins understands the risk of letting core assets spiral off on their own evolution trajectories: Assets that are allowed to leave the fold will essentially devolve back to the stovepipe development paradigm that product lines were meant to replace. The company faces the challenge of how to deal with this problem before it becomes unmanageable. Increased recognition and support of the product line approach by the U.S. government would help address this challenge and others needed in order to fully achieve the vision of a DoD-wide product line that supports all rotary-wing platforms.

Appendix A The Product Line Culture at Rockwell Collins

A mature product line organization is one that has achieved transition to a product line approach from a product-by-product approach for acquiring or developing software. The mature organization has established bounds on what constitutes the product line and has successfully introduced product line assets and a series of products built from those assets. Organizational elements are in place that have defined, developed, and fielded the product line. The relationship between the product line assets and systems within the product line is well understood and managed as new products are released. A defined and documented architecture guides the development process, both for assets and for new systems, and that architecture is maintained.

The product line organization applies an established process for fielding products in a product line and for sustaining product line assets. That process includes the following tasks:

- maintaining a shared understanding among stakeholders of the processes' goals. Stakeholders for the product line include sponsors, acquisition offices, asset developers, asset users, and product line product users.
- developing or acquiring product line assets and sustaining them throughout the life of the product line
- long-term planning for the product line and guidance for the development of specific product line outputs such as a developers' guide, business plan, architecture, and other assets
- managing the organizations that field the product line
- defining the role acquisition will play and solidifying the general acquisition approach. Acquisition involves procurement strategies for asset development, product development, and any needed contractual products and services.

In 1999, TAPO asked the SEI to conduct a series of in-depth interviews at Rockwell Collins in order to gauge the company's product line capabilities and the ability of the CMS architecture (the precursor to the CAAS) to be the basis for future evolution and commonality. What follows is an excerpt from that report. Although not all the findings were positive (among other things, we found a lack of formal product line processes and a strong oral [as opposed to documentation-based] culture), a picture of a conscious product line culture emerged.

1. Interview Summary

The first interview session of technical depth was with a group of senior designers called "project engineers." Project engineers are the technical authority for projects. They drew us a very detailed picture of the Flight Management Systems architecture (which we later learned was indeed one of Rockwell Collins' product lines). They also told us that projects get off the ground by first attempting to reuse, with as little change as possible, the architectural components from previous projects. In fact, the process starts with the software requirements specification (SRS); they have a "common" (what we would call a "core") SRS, off of which system-specific SRSs are written as deltas.[5] The project engineers also told us that they meet weekly to work out common requirements that they can feed jointly to the groups who maintain the major subsystems and associated software such as flight plans and guidance. We asked what prevented someone from picking up a component and changing it at will. The answer, they said, was that doing so was strongly discouraged. The functional area managers would not allow it, and the offending engineer would "have to eat by himself in the lunchroom." Beneath the levity of this comment, a picture of strong product line culture was beginning to emerge.

Subsequent interview groups confirmed that emergence. We learned that the project engineers' development environment (Rational's APEX) plays a critical role in the sustainment of the product line. For one thing, every project is open to every other project, so people can straightforwardly search for components that are candidates for reuse. Each component is stored along with the history of its usage and its requirements, test cases, and Ada specifications. APEX reveals what components depend on what other components, so the engineers can compare versions to quickly discern differences. We also learned that the functional area managers are the "keepers" of the product line. We asked to interview them—they were not originally on our schedule because we did not know they existed—and they revealed the whole product line picture.

About four and a half years ago, Rockwell Collins faced its own version of the CelsiusTech crisis.[6] Three major projects, which were expected to be won in a staggered fashion, came in simultaneously: KC-10, B-1B, and C-130. The company realized that it needed to exploit the commonality in all the systems, and the product line was born. But unlike with CelsiusTech, no champion from above ordered the architects into a locked closet. Rather, the functional area managers got together to work out procedures for exploiting the commonality. Projects apportioned the work and shared each other's components.

---

[5] We learned later that this common SRS is a work in progress, but nevertheless the vision is strong.

[6] For more information, see the technical report by Brownsword and Clements titled *A Case Study of Successful Product Line Development* [Brownsword 96]. CelsiusTech is a Swedish naval contractor that found itself in deep trouble after unexpectedly winning two major contracts simultaneously. Realizing that the company lacked the resources to fulfill the contracts at the same time, its only recourse was to build a single set of reusable software components (under the umbrella of a generic architecture) to meet both sets of requirements simultaneously—that is, CelsiusTech escaped disaster by turning to a product line approach.

Over time, the architectural picture that they showed us emerged. “Common subsystems” exist that almost never change from application to application. They are exemplified by the 1553 Bus Manager that handles putting messages on the bus and pulling them off for other subsystems. Other subsystems may change from project to project but in controlled ways. Subsystems have structures of their own; the “page” part (a *page* is a screen on the cockpit display device) changes often, but the state machine or controlling part seldom does. Each subsystem has an I/O part that communicates with other subsystems via the bus. The subsystem structure is reflected by groups of related Ada packages.

Today, the functional area managers control the design and evolution of the subsystems. We would call them the core asset group, but unlike other core asset groups, this one has no implementation duties. Rather, they are like senior designers who (a) approve or reject design changes requested by individual projects, (b) keep track of changes that are needed by more than one project, and (c) find projects that have the money to make the approved changes. (Sometimes changes are made using internal R&D (IR&D) money as well.) So this “core asset group” has the vision and the change authority, but changes are made by projects. (One way the tool helps them is that Apex automatically sends email to the cognizant functional area lead when a changed subsystem is checked in.) Each of these leads has a team of people who “work” for them and are farmed out to individual projects as needed. Each team represents a collection of expertise about a particular functional area.

We shared with this group our hypothesis that one difference between a mature product line organization and an immature one is that the mature organization sees its business as maintaining and nurturing the product line, whereas the immature one sees its business as turning out product via reuse. One of the managers immediately said, “That is no hypothesis” (meaning it was true). It is exactly how they view themselves—their primary responsibility is to the product line.

Improvements are continually made to the subsystems (which are their primary software core assets). The functional area managers are always looking for ways to improve the architecture and for projects that can use (and hence pay for) those improvements. Examples include (a) moving away from direct subroutine invocation as the collaboration mechanism and towards registration and (b) simplifying the pattern of dependencies among the subsystems, as evidenced by a “white noise” pattern of Ada “with” interactions.

Some of the conditions that work in the managers’ favor include

- a relatively small number of projects so far. The number of versions of any subsystem that a reuser has to browse through is no more than 15-20, and 6 is a much more likely number. Thus, they don’t have a runaway version-control problem yet.
- a domain in which the software architecture naturally mirrors the hardware/system architecture. Radios, identification friend or foe (IFF) devices, displays, navigational gear, and the like are all boxes that Rockwell Collins has made for years, and the software subsystems mirror them. Other software that doesn’t correspond precisely to a box (such as that for flight plans or aircraft configurations) rather naturally falls out.

- deep domain expertise, which is not a surprise

We presented a summary of our findings to the program manager and her technical director. In summary, we reported the following:

- The people we talked to were helpful and informative, and we had a series of high-quality conversations.
- It is clear that Rockwell Collins has a strong culture of product lines and architecture, which has emerged consistently. Everyone who drew the architecture drew the same picture. Everyone who told how the product line works told the same story.
- Rockwell Collins has an “oral” culture. If pressed, people could almost always point to documents that held requested information but couldn't always remember the title of the document or where it is located. They were much quicker to point to the right person to ask. A strong and formal mentoring process is in place—one that seems to be quite effective. There is also open communication. Developers are free to talk to their project engineers and “argue” with them about whether changes should be made. The project engineers will not hesitate to lobby the functional area managers for changes. The product line process is clear but not codified and not documented. The project engineers meet once a week—but apparently just because it’s their idea to do so. People do the right thing, but they do not do it because there is an authoritative document or process that tells them to.
- As far as we can tell, the architecture seems to succeed in
  - isolating processor dependencies. For example, they can port to a new processor in about eight days and have already ported to a PowerPC.
  - making it clear what’s changeable and what’s not, as evidenced by the distinction between common subsystems and application subsystems
  - isolating changes in the bus, by having a 1553 Manager component
  - isolating changes in message formats (via subsystems)
  - isolating changes in message types (by having a 1553 Manager configuration file)
  - making single-subsystem changes
  - isolating user interface (UI) changes (UI subsystem)
  - enabling the addition of new functionality
- The product line “process” appears to succeed in
  - discouraging ad hoc changes and “clone and own” (The owners of components are well known. Also, the Apex tool environment notifies owners via email about any changes that are made.)
  - keeping a small number of versions of components
  - encouraging (and achieving) high levels of reuse—Numbers as high as 94% were cited, and reuse in the 80% range seemed merely nominal.

- increasing component quality and reuse (by improving subsystems as time and resources permit)

- The CMS architecture might not be completely adoptable:
  - Function calls inhibit reusability.
  - “Cross-organizational” issues exist such as
    a. the amount of direct LRU-to-LRU invocation that exists among software components. Tight coupling among them renders them much less reusable.
    b. lack of documentation. This is crucial in cross-organizational reuse, since the organization picking up the components cannot simply walk down the hall to talk to their designers.
    c. The mentoring approach used successfully within Rockwell Collins won’t work for outsiders.

Appendix B Acronym List

API	application program interface
ARINC	Aeronautical Radio, Inc.
ASL	avionics system LAN
ASPP	Armament System Processor Panel
C2	command and control
C4I	command, control, communications, computers, and intelligence
CAAS	Common Avionics Architecture System
CDU	control display unit
CMS	Cockpit Management System
CORBA	Common Object Broker Request Architecture
CoRE	common reusable elements
COTS	commercial off-the-shelf
CSCI	Computer Software Configuration Item
DCU	data concentrator unit
DISR	DoD Information Technology Standards Registry
DoD	Department of Defense
FLIR	forward-looking infrared
GATM	Global Air Traffic Management
GPPU	general-purpose processing unit
GPS	global positioning system
HM	Health Monitor

IAC	Integrated Avionics Cockpit
IAS	Integrated Avionics System
IEEE	Institute of Electrical and Electronics Engineers
IFF	identification friend or foe
I/O	input/output
IR	infrared
IR&D	internal research and development
JSTARS	Joint Surveillance and Target Attack Radar System
LAN	local area network
LRU	line replaceable unit
MFD	multifunction display
MIL-STD	military standard
MMU	memory management unit
NDI	non-developmental item
NRE	nonrecurring engineering
OFP	Operational Flight Program
OSA	Open System Architecture
RCI	Rockwell Collins International
R&D	research and development
RF	radio frequency
RMA	Rate Monotonic Analysis
RSI	remote service interface
RSP	remote service provider
SAAM	Software Architecture Analysis Method
SEI	Software Engineering Institute

SLEP	Service Life Extension Program
SLOC	source lines of code
SOA	Special Operations Aviation
SOAR	Special Operations Aviation Regiment
SRS	software requirements specification
S/W	software
TAPO	Technical Applications Program Office
TCO	total cost of ownership
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
UI	user interface
VCT	Virtual Machine Configuration Table
WMD	weapons of mass destruction

References

URLs are valid as of the publication date of this document.

[Brownsword 96] Brownsword, Lisa & Clements, Paul. *A Case Study of Successful Product Line* Development (CMU/SEI-96-TR-016, ADA315802). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.016.html

[Clements 04] Clements, Paul & Northrop, Linda M. *A Framework for Software Product Line Practice, Version 4.2*. http://www.sei.cmu.edu/productlines/framework.html (2004).

[Clements 02] Clements, Paul & Northrop, Linda M. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

[Clements 99] Clements, Paul. "Software Product Lines: A New Paradigm for the New Century." *Crosstalk - The Journal of Defense Software Engineering 12,* 2 (February 1999): 20-22.

[Colucci 03] Colucci, Frank. "Avionics Upgrade Underway for Special Ops Helicopters." *National Defense 87,* 591 (February 2003): 24-26. http://www.nationaldefensemagazine.org/issues/2003/Feb/Avionics_Upgrade.htm (February 2003).

[Jones 99] Jones, L. G. *Product Line Acquisition in the DoD: The Promise, The Challenges* (CMU/SEI-99-TN-011, ADA373184). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. http://www.sei.cmu.edu/publications/documents/99.reports/99tn011/99tn011abstract.html

[Northrop 04] Northrop, Linda M. *Software Product Line Adoption Roadmap* (CMU/SEI-2004-TR-022). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tr022.html

[Northrop 02] Northrop, Linda M. “SEI's Software Product Line Tenets.” *IEEE Software 19*, 4 (July/August 2002): 32-40.

[Rockwell Collins 05] Rockwell Collins. *Rockwell Collins Develops Key Technology Solutions to Meet Military's Transformation Objectives.* http://www.rockwellcollins.com/news/background /page3072.html (2005).

[Thompson 05] Thompson, Brian L. & Ward, David. “Common Avionics Architecture System (CAA) Flight Test Program,” 123-137. Society of Experimental Test Pilots 49th Symposium & Banquet. Anaheim, CA, September 28 – October 1, 2005. Lancaster, CA: Society for Experimental Test Pilots, 2005.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2005	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study	FA8721-05-C-0003

6. AUTHOR(S)

Paul Clements, John Bergey

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2005-TR-019

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2005-019

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This report is one in a series of Carnegie Mellon® Software Engineering Institute case studies of organizations that have adopted a software product line approach for developing a family of software-intensive systems. The U.S. Army's Technical Applications Program Office (TAPO) has adopted a product line approach for the avionics software used for the Army's special operations helicopters. That software is based on Rockwell Collins' Common Avionics Architecture System (CAAS). The product line has evolved beyond its original scope and is now being adopted to include other Army aviation platforms such as cargo and utility helicopters.

This case study describes the acquisition context and organizations involved in the product line, the history behind the development and evolution of the product line, its application to the mission of the Army's special operations helicopters, the Army's motivation for adopting a product line, specifics of the product line approach, and the underlying CAAS system and software architecture. The case study also highlights the software product line accomplishments, examines the results and lessons learned from TAPO's and Rockwell Collins' perspective, and discusses future considerations.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software product line, product line case study, product lines in acquisition	62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL