Capability Driven Architecture (CDA)
an Approach to
Rapid Platform Integration

Written by: **Tucson Embedded Systems, Inc.**
January 2012

Updated for Publication**:**

June 2021

Executive Summary

CDA presents a unique approach for integrating devices in that it combines a three-pronged solution for integration. These are 1) an interface development process (i.e., the Super API), 2) a toolset supporting that process, and 3) a software architecture supporting the rapid prototyping effort to demonstrate the "Plug and Play" capability. The CDA toolset is utilized to manage the complex and large amount of interface data for each device. Without the toolset and formal process, the volume and complexity of the data would quickly become unmanageable. Additionally, the toolset's support of airworthy development through requirement management, design, software development, and traceability increases confidence of the resultant system.

CDA is a proven technology demonstrated through a number of related Government and commercial programs including: Army communication systems integration, independent validation and verification as a first time third-party user, and application to a number of technology and development efforts. CDA has been shown to reduce cost through simplified interfaces, auto-generation of software, tests, and documentation and software reuse. It reduces schedule by allowing concurrent software design, development, integration, testing, and responsiveness to evolving or changing requirements by planning for them. Finally, it reduces risk by making cost and schedule more predictable and controllable, enabling detailed visualization of the entire process, and management of all data artifacts from initial requirements through design, development and test and lifecycle sustainment.

The inherent ability of CDA to provide a complete "cradle to grave" reusable software solution for device integration in an FAA and military airworthy environment is unique. The ability to reduce cost, schedule, and risk, and to provide an effective solution to the multiple integration issue meets the rapid integration objective of many programs.

Table of Contents

List of Figures

1.1 Questions Pertaining to this Document

Any questions on or pertaining to this document should be sent to the attention of Mr. Stephen Simi, Tucson Embedded Systems, Inc. Program Manager at StephenS@TucsonEmbedded.com.

1.2 Terms

Super API	An Application Programming Interface providing an overarching, high-level platform access to the domains functionality.
API	Application Programming Interface
SA	Situational Awareness
MOSA	Modular Open Systems Approach: An integrated business and technical strategy that: – provides an enabling environment – employs a modular design – defines key interfaces – uses widely supported, open standards that are published and maintained by a recognized industry standards organization – uses certified conformant products
Product Suite	The complete HW/SW implementation of rapid integration technologies.
OS	Operating System: The Operating System that the Product Suite operates within.
RTOS	Real-Time Operating System: Refers to a special type of operating systems typically used in embedded systems requiring deterministic execution environments. This is sometimes called *hard* real-time.

Table 1: Terms

2. CDA Description

2.1 Overview

Capability Driven Architecture (CDA)[TES Patent [1]] is an architecture initially developed to achieve rapid development and portability of software based capabilities across multiple avionics platforms. This is achieved with a three-pronged solution (Figure 1) of providing a process for developing hardware independent functional groupings called capabilities, a toolset for supporting the CDA process, and airworthy platform agnostic software architecture. The process and toolset's support the complete software lifecycle including: capability development, management and traceability of application programming interfaces, verification of the software components, and airworthy artifact and traceability needs.

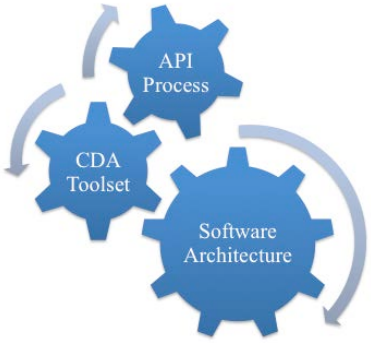

Figure 1: CDA Three-Pronged Approach

2.1.1 CDA Background and Purpose

CDA was primarily built to support the integration of common devices upon dissimilar airworthy platforms (as illustrated in Figure 2). CDA is more than a code generation tool, or a test and verification system. It is a design methodology that promotes software reuse by abstracting platform environment and device capabilities into common or standard interfaces. This methodology is supported by open platform tools and is backed by a rich history of safety critical aviation flight software systems. All of the software built or generated under the CDA system is developed upon of a base level of software known as the operating environment or OE. This set of code allows for the deterministic capabilities in a cross-platform development package. While currently actively supporting both Windows 32-bit and 64-bit, Linux 32-bit and 64-bit, and ARM-based processor systems, TES has also successfully shown that CDA-based applications run on a host of Real-Time Operating Systems (RTOS): LinuxWork's LynxOS, Greenhill's Integrity, and Wind River's VxWorks; on varied embedded processors such as PowerPC and Intel architectures.

By supporting such a wide variety of operating systems[2], TES and the CDA concepts can easily support the platforms that run these systems. CDA has been developed to provide the safety critical systems needed to fly and service various rotorcraft such as: the OH-58D Kiowa Warrior, UH-60L 60M Black Hawk, AH-64A/D Longbow Apache, and CH-47F Chinook.

[1] TES' CDA is a patented technology. All CDA by-products developed are provided to the Government and marked "Government Purpose Rights" or "Unlimited Rights".

[2] Specific architecture platforms and RTOS versions are identified in the System and Software Requirements Specification.

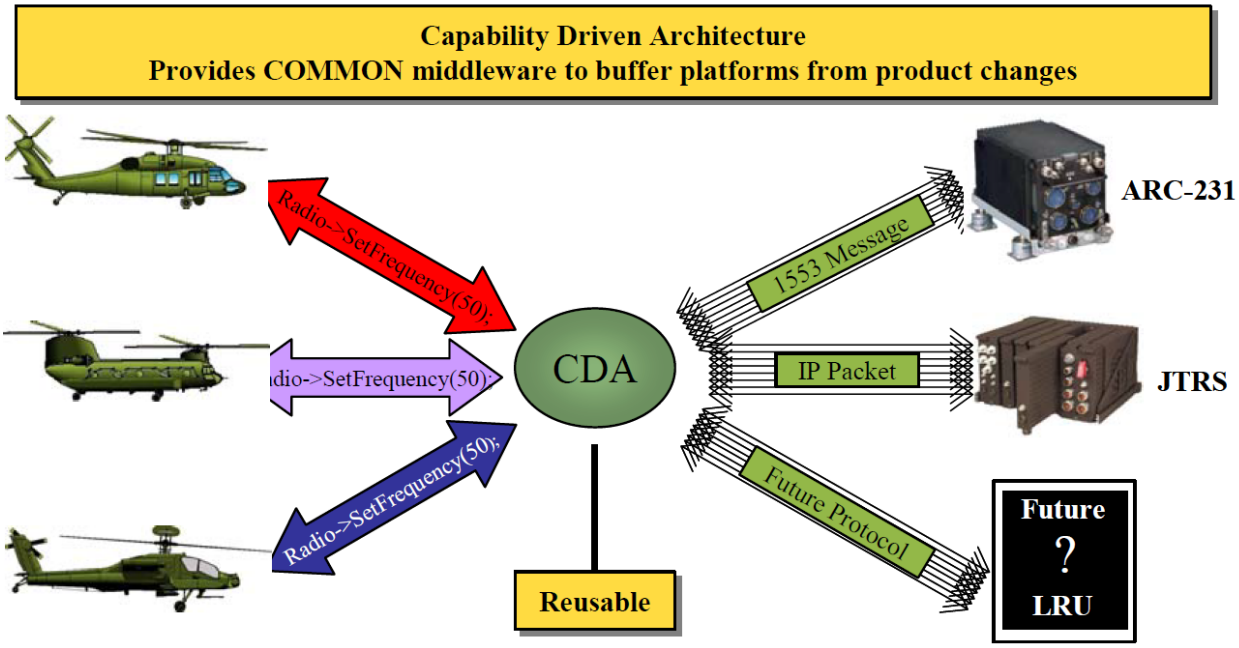

Figure 2: CDA provides common middleware to buffer platforms from product changes

2.1.2 Model-Driven and Domain-Specific Approach – Applied to Mission & Safety Critical

CDA is a software architecture that combines the Model Driven Architecture (MDA) approach with a Domain Specific Model approach through software design and implementation. CDA focuses on creating a platform-independent model of the software system. This independence is created by using a model that relies on automated tools to translate that model to software rather than being developed to a specific platform RTOS. CDA extends beyond the MDA approach by emphasizing the development and integration of "capabilities" rather than on the specific systems, sub-systems, or hardware.

A formal process is used to identify the inherent capabilities of the system and subsystems. This process is iterative, allowing a "churn" of capability sets so that commonalities and taxonomy of middleware are identified for the simplification of cross-platform integration and the benefit reuse. This process is described further below.

Another major design feature of CDA is that it applies the favorable aspects of MDA to mission and safety-critical applications. Typically, these applications have hard and soft real-time system requirements with strict traceability needs. CDA has been architected from the ground-up to meet strict airworthy requirements and has been demonstrated for reusing mission and safety critical software components [FAA's AC 20-148]. CDA is an open standards-based architecture for building highly reliable applications of various sizes in both local and highly distributed environments. Unique to CDA and its process is its ability to integrate and deploy new and legacy hardware and software capabilities as one holistic reusable environment onto various platforms in a mission critical system.

2.2 API Creation

2.2.1 API Creation Process

The CDA process is a combination of bottom-up and top-down approaches where the input to the process is the low-level interface documents and the system requirements. These low-level documents are imported into the CDA toolset (described in Figure 3), and reside in a database for an abstraction of the data resulting in a top-down open and commonality-based design. The high-level system requirements are also entered into the toolset. The remaining process fills in the gaps between the system requirements and the low-level ICDs.

2.2.2 Functional Abstraction

The functional abstraction analysis process is iterative in nature. It is used to define standard interfaces and categorize the underlying control code for the capability.

The primary idea behind the process is that by documenting the detailed interfaces, bubbling those interfaces up into their primary functions, and then bubbling up those functions into capabilities provides a process by which a complete capability interface can be defined.

The input into the CDA process is low-level Interface Control Documents (ICDs) for defining application-level interfaces, and protocol and operating system specification documents for operating environment interfaces. The combination of these two high and low-level interfaces promotes CDA's ability to rapidly integrate common and dissimilar capabilities and devices on dissimilar platforms. This abstraction process is depicted as follows.

The process works for both the high-level application and device interfaces as well as the low-level operating environment (OE) interface. The operating environment API allows all CDA implementations to use operating system (OS) services such as threads, mutual exclusion, file operations, and timers; yet, it isolates the implementations from the specifics of every operating system on which CDA executes.

It is the CDA-OE that allows CDA applications to be supported on all of the major operating systems such as Windows, Linux, and RTOSs (such as VxWorks, LynxOs, and Integrity). Adding additional platform support in CDA-OE is a straightforward effort.

This process of iteratively refining the interfaces can be a difficult job to manage by hand. This is why a software toolset, aptly named CDA, was developed.

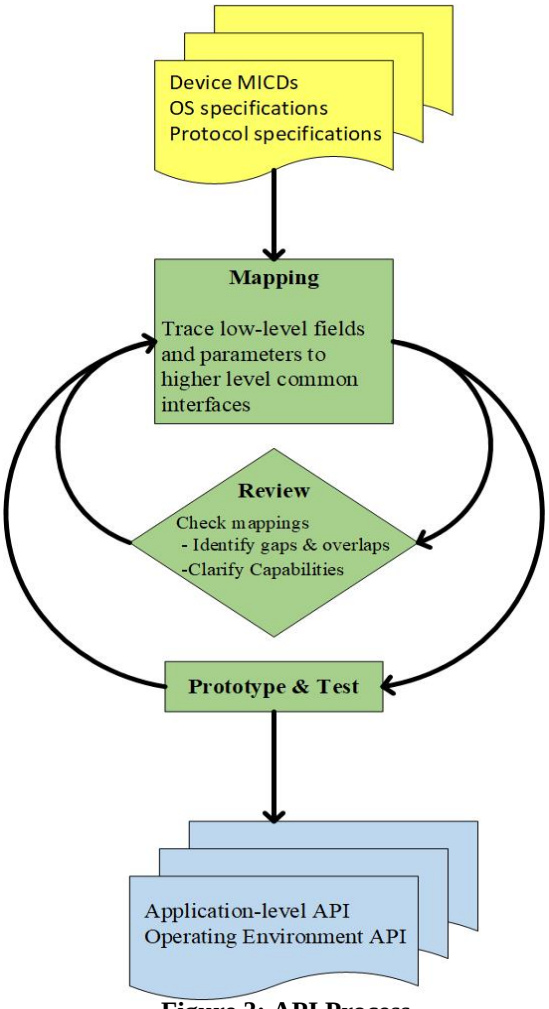

Figure 3: API Process

For example, the results expected on the JTRS AMF-SA Radio Control API program are one common API used to control three Army radios, 11 waveforms, simplify the impact of changes to the Platforms, reduce program risk, and program integration costs and schedule. It is expected that TES processes the 1,700 pages of the JTRS AMF-SA program Radio ICDs, MIB files and waveforms, and develop one Radio Control API. The process and result is illustrated in Figure 4 below.

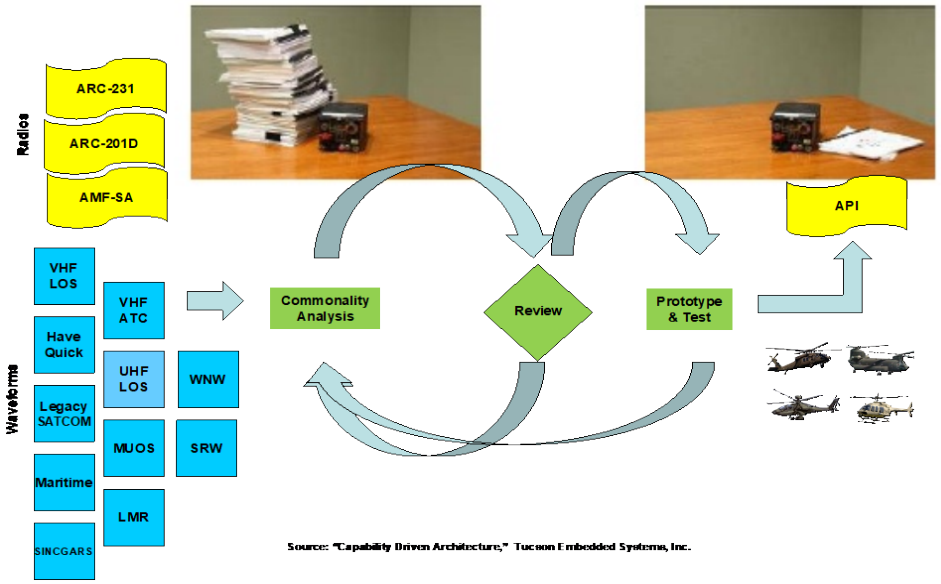

Figure 4: LRU API Abstraction and Trace Process

Once the API is defined, the interface requirements document can be auto-generated from the CDA toolset. Additionally, the design of the interface, the design of the code, and the design of the configuration data (which implements the data caching) can be auto-generated. Furthermore, test cases and test procedures can also be generated from CDA toolset. Also, the CDA toolset can auto-generate software code, test cases and procedures, and documentation artifacts. The code modules currently implemented are the C++ messaging code, CORBA IDL, ONC-RPC IDL, Java messaging, and a Java GUI display.

2.2.3 Design and Lifecycle Artifacts

The CDA process and toolset supports each stage of the development phase's lifecycle. Listed below is a description of the application of CDA at each phase and the corresponding benefits:

- Requirements Analysis – Generation of requirements from the Model significantly reduces errors and provides a more complete and correct set of requirements which trace directly to the detailed specifications and code.
- High Level Design – Generation of the High-Level design with accommodations for application-unique design, such as tasking/processes. The CDA model supports reuse of the high-level design of each process in that many modules are reused; this includes all of the documents, code, unit testing, and integration testing.
- Detailed Specifications – The detailed specification can be completely generated from the CDA model and toolset. This includes tracing to the requirements, as well as tracing to the raw ICD data and APIs.
- Coding – All of the code for handling publish-subscribe data is auto-generated. This eliminates any possible hand-coding errors. It simply works. Also, much of the coding style and significant modularity of the software is driven by the CDA process. Therefore, the programmer has to implement only that code which is difficult; they do not spend time on tedious coding which is very error prone. Instead, they spend their time on the actual application.
- Unit Testing – The modularity of the software developed through CDA give the ability to easily test the modules, as well as the opportunity to auto-generate test cases and procedures for each module. This method provides a greatly reduced unit testing effort.

- Integration Testing – The publish-subscribe interface provides a well-defined specification for integration testing and simulation. It provides the ability to simulate those inputs that are difficult to provide in a stimulated environment. This combined with the Programmable Control Test Station's (PCTS) ability to stimulate those inputs/outputs gives a much more complete integration environment.
- Operational Testing – Testing at this level must still occur, but is significantly simplified due to the correctness and completeness of the modules developed with the CDA process. In addition, the ability to rapidly prototype a functional application early on in the development phase provides important operational feedback, which is invaluable in determining the operational requirements. This increases the likelihood that the resulting system functions as needed.
- Artifact Generation and Lifecycle Support through Airworthiness – The CDA toolset has the ability to auto-generate lifecycle artifacts. TES had worked with Army representatives from the Aviation & Missile Research, Development, and Engineering Center (AMRDEC) Software Engineering Directorate (SED) to confirm that the results were suitable as supporting artifacts for Airworthiness Qualification Substantiation Records as defined in AR 70-62.

The lifecycle artifacts include those that are required to support airworthiness certification processes. These are planning, requirements, testing documents, and many have been designed for reuse, per AC-20-148. The table below lists the artifacts required for AWR and identifies those that can be reused across other platforms.

Software Artifacts	Reusable
Plan for Software Aspects of Certification	Yes
Software Development Plan	Yes
Software Configuration Management Plan	Yes
Software Quality Assurance Plan	Yes
Platform Functional Requirements	Unique per Platform
R2C2 Software Requirements Specification	Yes
Requirements Traceability Matrix	Yes
Software Design Description	Yes
API Specification	Yes
Software Test Plan	Yes
Software Test Report	Unique per Platform
Software Structural Coverage Analysis / Testing	Yes
Software Verification Cases, Scripts, & Procedures	Yes
Software Accomplishment Summary	Yes
Safety Assessment Report	Unique per Platform
Software Problem/Change Reports	Unique per Platform
Software Version Description	TBD
Integrator's Users Guide	Yes
R2C2 Source Code	Yes

Figure 5: Reusable Artifacts

Requirements (SSS/SRS/IRS), Design (SDD/IDD), Interface Control Document (ICD), Software/System Verification and Procedures (SVCP) include full traceability from requirements to design, implementation and verification. Through the auto generation of documents, code, test, and simulation, the programmer workload is significantly reduced. In effect, the "busy work" of programming and testing is performed by development of the importers, code generators, and artifact generators.

2.2.4 Testing and Verification

The process of creating fully defined and published APIs provides significant benefits for testing and verification. These benefits are fully defined test points, early test development, auto-generation of test cases and procedures, full traceability and documentation. First, the APIs defined in an SRS or IRS provide specific interfaces that can be fully tested and verified. Second, since the API definition is performed early in the development effort, test cases and procedures can be developed early as well. Third, since the full API is stored in a computer-readable format, a significant amount of test cases and procedures can be auto-generated. Lastly, an integrated test station can utilize the auto-generated test vectors, test procedures and the developed device API to provide a hardware-in-the-loop test environment with traceability and documentation. In our experience, testing and verification is one of the more significant savings opportunities derived from the CDA process as we continue to expand the capabilities of the test auto-generation toolset.

2.3 CDA Toolset

The CDA toolset is a multi-platform set of Eclipse plug-ins developed to manage all of the aspects of CDA application development. Eclipse is an open source industry standard for Software Integrated Developers Environments. It provides significant functionality out-of-box for programming in Java, C/C++, and many other languages. In addition, Configuration management, software review, and error tracking tools are available, as well as many other plug-ins too numerous to list here.

CDA tools are built to support various roles or perspectives based on the different software development lifecycle needs. These perspectives match the normal software development lifecycle, but also include other views beyond just software development: CONOPS, Modeling and Simulation, Situational Awareness in 2D and 3D, Platform independent models, Platform Specific Models, and more.

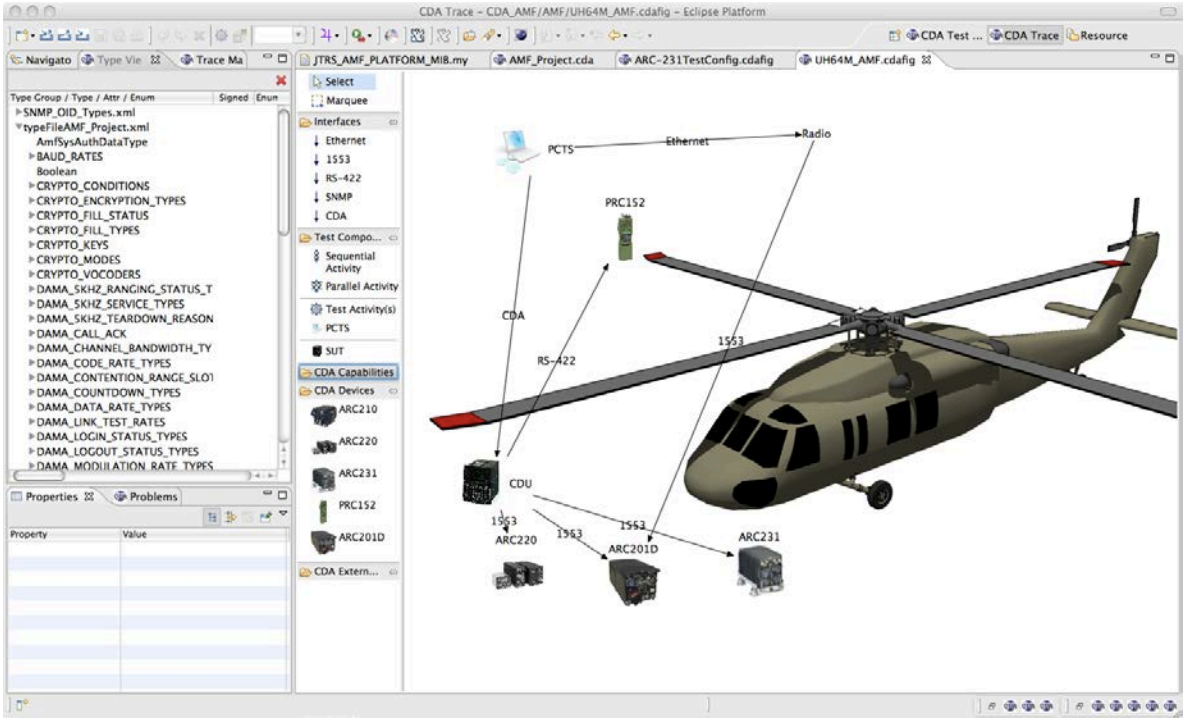

Figure 6: CDA Example Platform Integration View

The developer first starts with the import of ICDs, MICDs, or APIs. Once available, Systems requirements are entered into the toolset. At this point, the CONOPS and Systems level diagrams can be developed to provide a model of the Concept of Operations and the Platform Integration View (Figure 6).

The toolset is a combination of tools that work together to provide a complete requirements, design, development, and integration package. These tools are listed below:

- Import Tool – used to read MICDs, ICDs, protocol and operating system specification documents
- Requirements Viewer and Trace Tools
- Diagram Editor – used to input the CONOPS, Platform Design, Device View, Platform Integration Views, Simulation Scenarios, and Test Environments
- Capability Viewer – used to generate Unified Modeling Language (UML®) and other design diagrams
- Comparison Tool – used to generate variance reports between capabilities
- Capability, API, and ICD Trace Tools – used to develop common APIs and perform requirements tracing through the process
- Artifact Generator – used to generate automated test scripts, prototype control code, and lifecycle documentation

The CDA tools are grouped to support the CDA process with five management capabilities. These toolset capabilities allow engineers to design application-level and OE-level APIs. The capabilities include:

- Interface Management
- Requirements Management
- Capabilities-Based Design
- Software Development
- Test Development and Execution

2.3.1 Interface Management

The CDA toolset provides a unique set of tools for the importation of many device interfaces: the creation and mapping of abstracted Application Programming Interfaces (APIs) and tracing to the device interfaces. The importer tools provide the entry of device and operating data into the CDA database.

For example, if the developer is implementing an interface to a family of devices such as GPSs, the device ICDs can be imported into the CDA toolset. Once the device ICDs are imported, the message fields are traced to the API/service interface. The API provides a simple function call such as “getLocation()”, and link to set ???Device ICD’s function “currLat()”, “currLong()”. The “getLocation()” now returns a standardized location regardless of the device connected to the platform. Once this relatively simple process is accomplished, the interface code, design artifacts, and TCs/TPs can be auto-generated.

2.3.2 Requirements Management

A tool for basic requirements data entry, traceability and tracking is provided to developers to simplify the management of requirements. In addition, it provides developers the ability to quickly access, update, and trace to application requirements. The following capabilities are provided:

- Requirements Entry
- Full requirements traceability

- DOORS® export/import
- Document Generation
- Traceability Matrix Generation

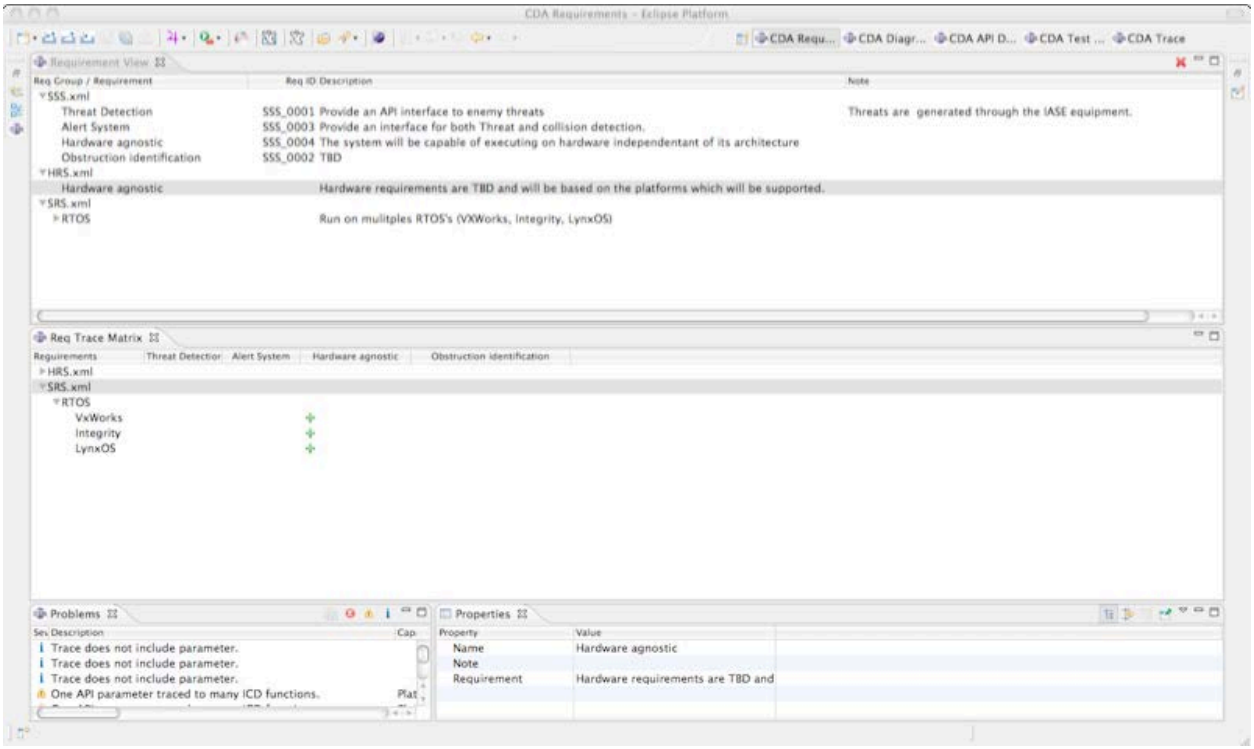

Figure 7: CDA Toolset Requirements Perspective

2.3.3 Capabilities-Based Design

The Capability design tools provide a visual method for mapping to requirements, platforms, and device interfaces. These tools support various design views in the user's vernacular or user's domain. These are the ICD view, Concept of Operations (CONOPS) view, Platform Integration view, and device integration view. The concept of visualizing the design from the various users' domain is very powerful. For example, platform systems engineers can view the design from a platform integration level, and they can drill down to the device perspectives (Figure 7). This provides an interface to increase the understanding of the platform as a whole while providing quick access to the details of the system (Figure 8).

Since the complete details for the platforms software integration are stored in machine-readable format, documentation can be auto generated in part or in whole. This includes such documentation as the SSS, SRS/IRS, SDD/IDD, and UML® formats.

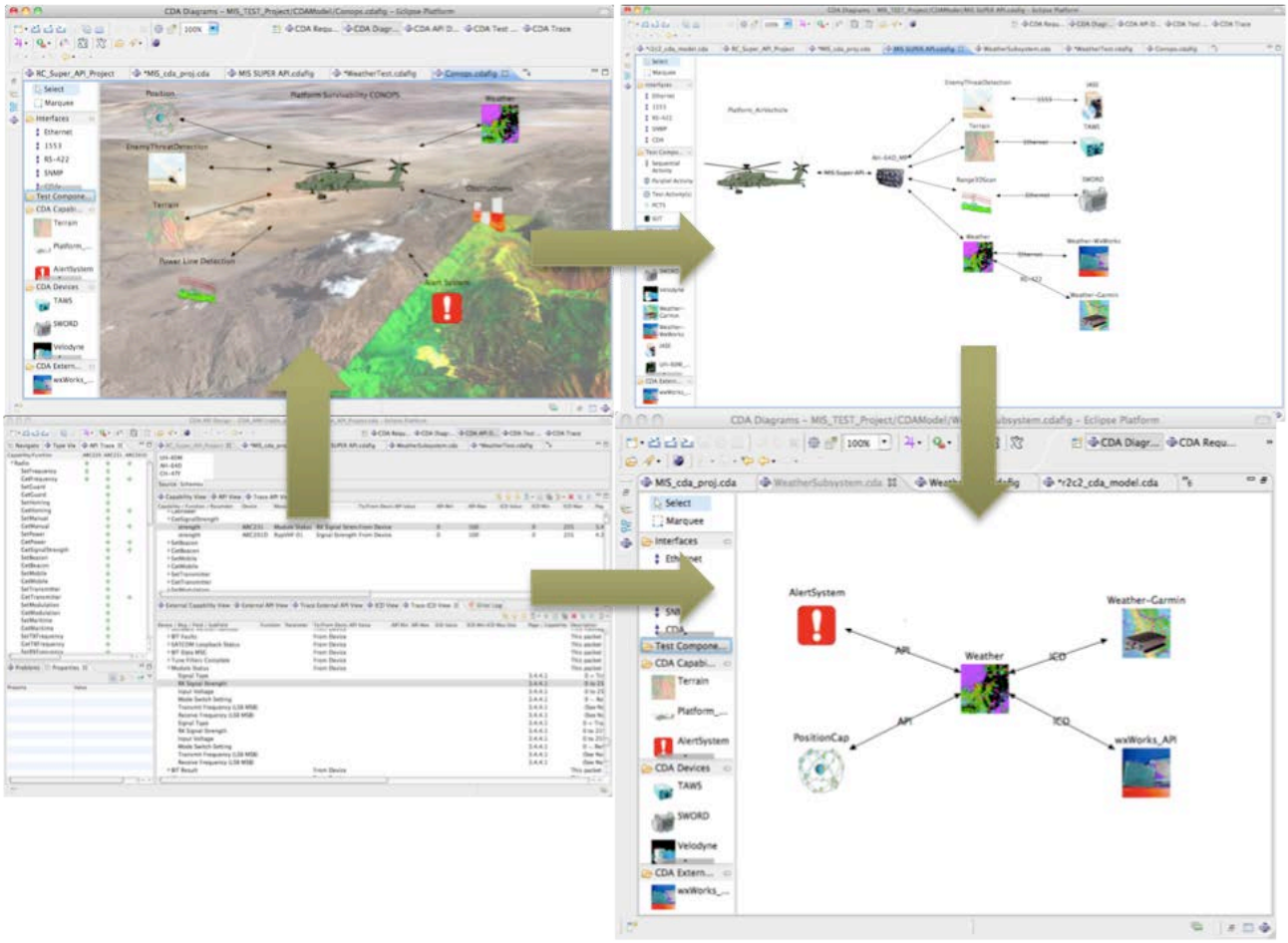

Figure 8: Capability Design Views

In addition, during design and development, the traceability between the component pieces of the design are maintained at all levels. This provides the additional ability for the developer to manage the traceability at all levels.

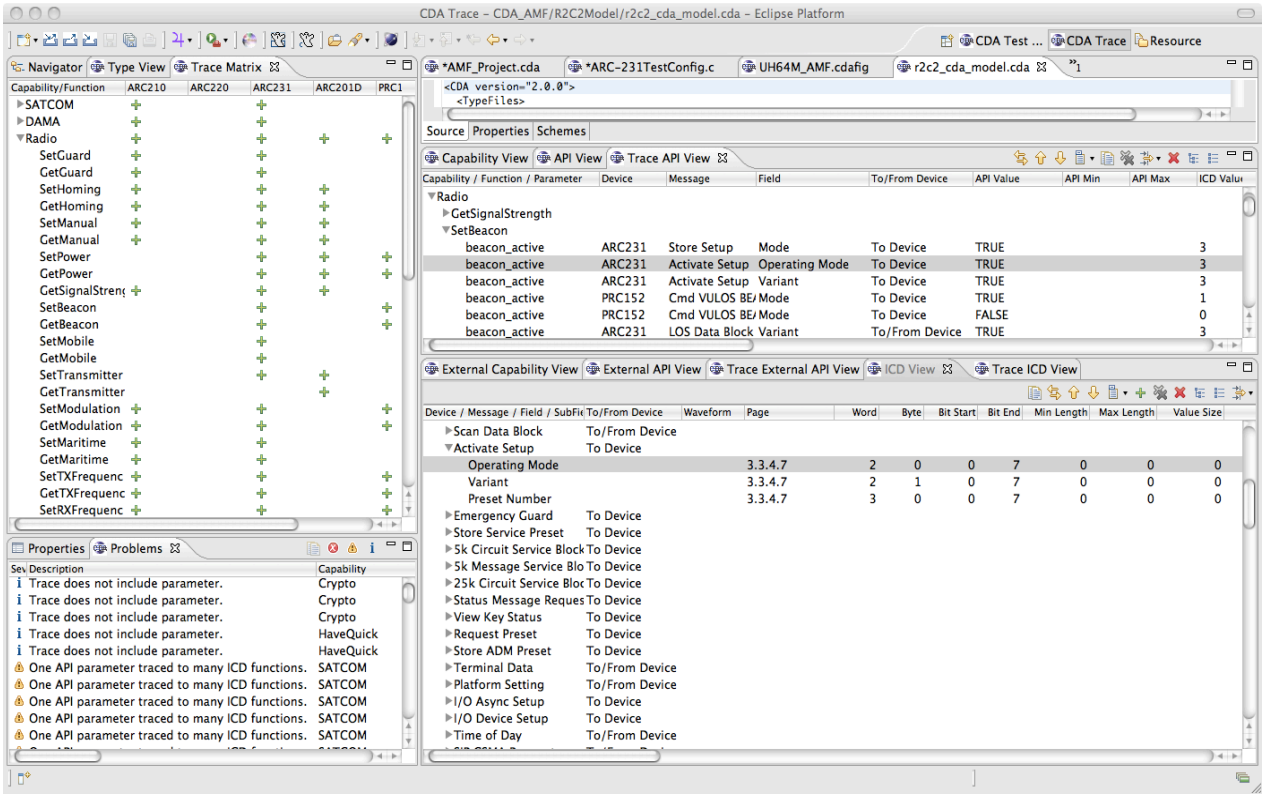

Figure 9: API to ICD Tracing View

The toolset can also support DOORS® integration through the Eclipse import and export mechanism utilized for ICD import and export. This allows the developer to have quick access to full bi-directional traceability from requirements, to design and to test cases and procedures to test results.

2.3.4 Software Development

As stated above, the CDA toolset is an Integrated Software Development Environment based on industry standard Eclipse. It is this integration with the standard Eclipse environment that provides a complete software development environment (Figure 10). The following tools are provided for the software development team:

- Integrated Team Development Tools
    - Build Management
    - Version Management
    - Bug Tracking
    - Code Reviews
- Code generation
- Test generation, execution, and results
- Traceability Support to Requirements and Design

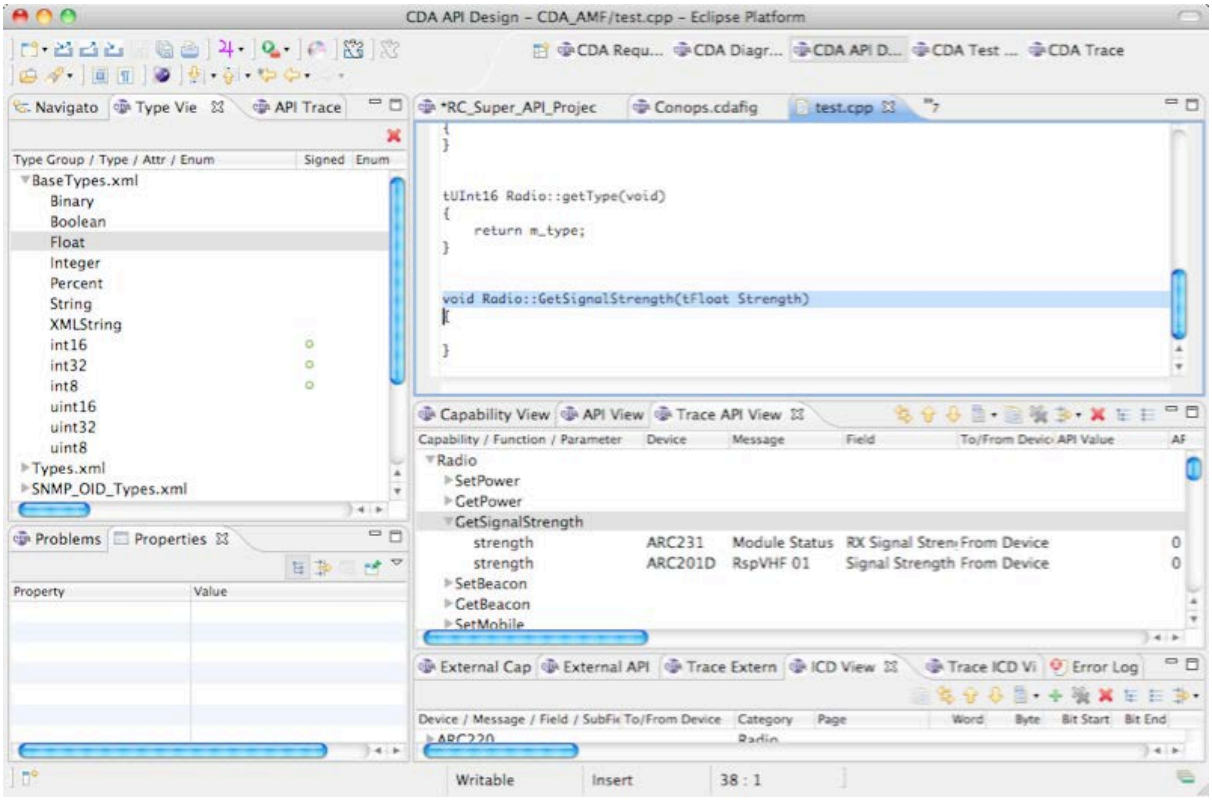

Figure 10: Software Development Environment

2.3.5 Test Development and Execution

The Programmable Control Test Station (PCTS) was developed for automated testing in mission and safety critical systems. PCTS has since been integrated as part of the CDA toolset (Figure 11). PCTS includes the following functions:

- Test Case & Test Procedure Development
- Execute Batched Test Scripts against stimulated/simulated devices
- Visualization of Recorded Test Results
- Supports Traceability of Test Cases and procedures to Requirements and test results to test procedures

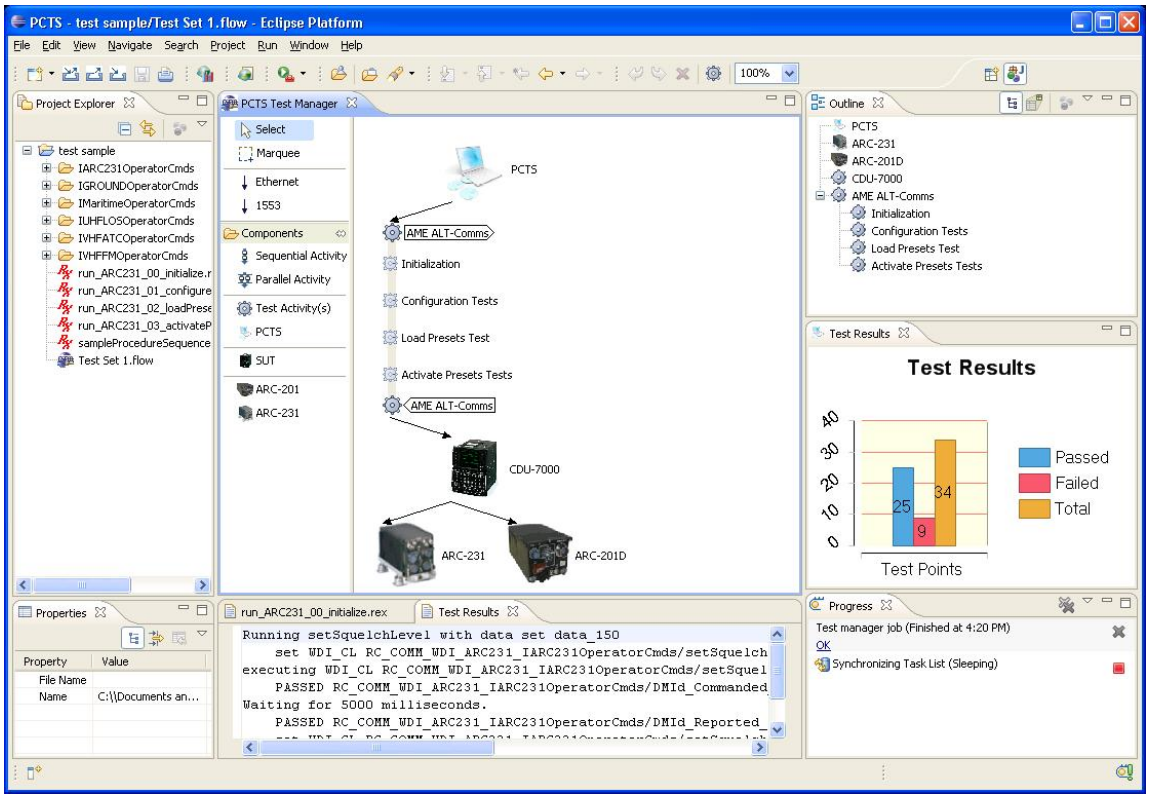

Figure 11: Test Environment Perspective

2.4 Software Architecture

The key to common reusable software is being able to isolate the software from the differences between the various platforms and devices. 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 devices, the platform applications become device-independent and reusable with different devices.

For these reasons, CDA replaces platform-unique code with a structure of three layers of abstraction: an Operating Environment (OE) abstraction layer, a software framework for developing CDA applications, and an interface abstraction layer.

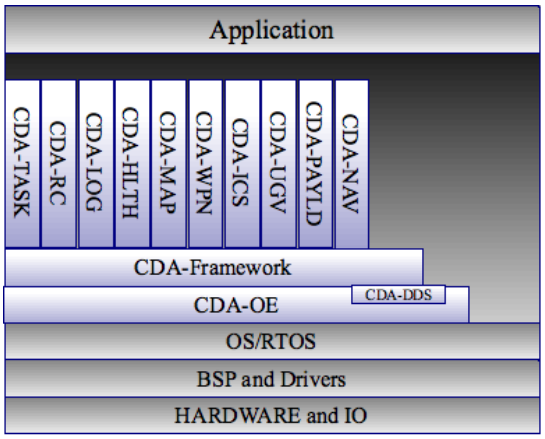

Figure 12: CDA Abstraction Layers

Through these layers of abstraction (Figure 12), CDA effectively reduces the software development 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 interfaces/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.

2.4.1 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 standard interfaces defined in the OE abstraction layer (Figure 13).

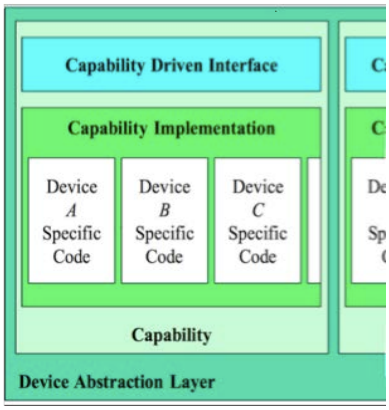

Figure 13: Operating Environment Abstraction Layer

2.4.2 Service Abstraction Layer

The service 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 and produce positioning data such as GPS or Inertial Measurement Units are in the Position Capability.

At the top level of a Capability is the capability driven interface. 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 does not need to change when replacing one device with a different device that has the similar Capability. In fact, this feature has been proven through the ability to switch from a Trimble GPS position to a Honeywell VNU, or a BAE IMU, without affecting any applications other than the CDA-NAV component.

This abstraction is implemented in CDA; yet, it 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.

2.4.3 Data Messaging and Synchronization

The messaging system utilized by CDA is the Capability Driven Architecture Data Distribution Service (CDA-DDS). CDA-DDS provides the backbone and foundation for all of the interfaces between devices and platform applications in the system. CDA minimizes the data coupling between software components and provides support for various languages (C++, Ada, and Java), as well as a host of network protocols (TCP/IP, UDP, MIL-STD-1553, RS232/422/485, and ZigBee 802.15.4).

CDA-DDS was modeled after the OMG DDS. CDA-DDS also provides some significant enhancements to OMG DDS to support hard real-time systems and for low-bandwidth and disconnected network operation.

CDA-DDS provides significant benefits for application developers by providing a rich set of tools to model the data that is transferred between components and for monitoring and debugging your application. Application data can be configured rigidly for hard real-time systems (through code generation) or “on the fly” by the application.

2.5 Other Related Efforts

2.5.1 CDA Component Reuse and Demonstration Efforts

The key to common and reusable software is being able to isolate the software from the differences between the various platforms and devices. 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 devices, the platform applications become device-independent and reusable with different devices.

CDA-RC was demonstrated in 2006 to PM-AME as a viable reusable architecture for Army Aviation radio control. The CDA process is now being reused on the JTRS AMF-SA program to design the Radio Control API for the common control of the AMF-SA, AN/ARC-231, and AN/ARC-201D radios. Collectively, the Radio Control API should control 3 radios and 11 waveforms. These AMF-SA Radio Control API efforts are to be demonstrated 2QTR 2011 to PM-AME. CDA process and verifications capabilities are being used to perform IV&V on PM-AME Alt-Comms suite of reusable software. TES is testing each engineering release and identifying operational functional issues early in the lifecycle, thereby reducing program integration costs and making the aircraft safer.

2.5.2 AME Alt-Comms Radio Control IV&V – WDI IDK

The CDA toolset is being utilized to perform an independent verification of Army Aviation PM-AME Alt-Comms Well Defined Interface (WDI) Integrators Developers Kit (IDK). CDA is being used to import the WDI IDK C++ header files into the toolkit’s ICD database. This data is then used to create a test-harness to provide a bridge for the CDA PCTS. The database is also used to generate test cases and test procedures that are executed with the PCTS.

2.5.3 JTRS AMF-SA API Design and Prototype

The CDA toolset is being utilized to develop a Super API for Radio control software for the JTRS AMF-SA radio, the AN/ARC-231 radio, and the AN/ARC-201D radio. CDA import plug-ins have been developed to import all of the radio ICDs and the AMF MIBS for SRW and WNW. Once imported, the ICDs are being run through the API development process to create a set of functional interfaces for these radios. The interfaces includes full traceability to all of the message fields and SNMP MIB variables. The resultant API is documented in both human readable and digital formats as an Interface Requirements Specification (IRS), XML data format, and UML/XMI.

2.5.4 Commercial Applications

CDA is being utilized in many different areas of engineering including:

- Wireless sensor system for Mining asset tracking
- Formal verification for Commercial aviation O2 System for DO-178B Level B
- University of Arizona UGV Capstone effort