Architecture-Driven Assurance for Safety-Critical Avionic Systems

Dan Bliek
Principal Systems Engineer
Rockwell Collins
Cedar Rapids, IA, USA

ABSTRACT
As the growth in complexity of avionic systems continues, development costs and schedules have correspondingly increased dramatically. Systems engineering tools and methods have not sufficiently adapted to the demands of today’s complex systems and as a result avionic systems typically field late, over budget, and often with reduced capabilities. The status quo for systems and software engineering tools, methods and processes is no longer affordable.

Architecture-Driven Assurance is model-based systems engineering (MBSE) development approach for constructing reliable and secure systems using engineering models. The objective of the architecture-driven assurance methodology is to ensure that cyber-resilient, functionally correct, verifiably safe components can be rapidly developed, integrated and verified. The approach integrates compositional reasoning into the engineering workflow utilizing system models and formalized descriptions of system behaviors. The models are translated into a form that can be interpreted by powerful formal methods based general-purpose analysis engines. Rockwell Collins has created an integrated development environment that supports the architecture-driven assurance concepts integrated into a systems engineering workflow.

INTRODUCTION
As the growth in complexity of avionic systems continues, correspondingly development costs and schedules have increased dramatically (see Figure 1). Recent major development programs such as F-35 continue to provide ample evidence that the status quo for systems and software engineering tools, methods and processes are no longer affordable. Systems engineering tools and methods have not sufficiently adapted to the demands of today’s complex systems. As a result, avionic systems typically field late over budget, and often with reduced capabilities.

In addition to avionic systems becoming larger and more complex, there are new integration and verification challenges driven by an increased emphasis on:

- Dramatic reductions in system fielding times (years to months, months to days)
- Operational flexibility and technical superiority with rapidly composable heterogeneous systems
- Increased focus on efficient integration and verification driven by open systems architectures (OSA) initiatives

This paper presents Rockwell Collins’ Architecture-Driven Assurance approach to model-based systems engineering (MBSE), that uses compositional reasoning and verification (reference 1). Architecture-driven assurance is the combination of system structure, or architecture, with formally expressed system behaviors. This method requires tools that enable:

1. Modeling the system architecture using a standard notation that is semantically rich and precise.
2. A means of formally expressing required system behaviors to enable analysis plus the ability to attach the behaviors to the system architecture.
3. Techniques that allow for the division of the analysis tasks, tied to the system architecture hierarchy, into manageable, reusable elements.

Over the past several years, the Rockwell Collins Advanced Technology Center (ATC) has performed research on programs focused on the application of formal methods and architectural analysis at all levels of system development. The tools and methods have been, and are being, refined through a combination of technology development and technology application programs. The technology development programs create or enhance MBSE capabilities. The technology application programs apply architecture-driven assurance methods and tools to relevant avionic systems to validate benefits and identify gaps. An integral part of the technology application programs has been to create and refine an engineering workflow that leverages architecture-driven assurance to enable a pragmatic use of formal methods in the defining, designing and implementation of safety-critical avionic systems.

**FUNDAMENTALS OF ARCHITECTURE-DRIVEN ASSURANCE**

Meeting the objective of rapidly developing, integrating and verifying functionally correct and provably safe components requires an assurance approach that is capable of meeting five fundamental precepts:

- Requirements are provably correct
- Architecture model is correct
- Components are correct
- System execution conforms to the model
- System implementation corresponds to the model

The fundamental precepts are further described in the following subsections.

**Requirements Are Provably Correct**

Functional properties constitute a set of behaviors that the system is intended to provide. Expressing the functional properties in a formal analyzable language that also follows the description of the system architecture is important to managing the scalability challenge by dividing the analysis tasks into manageable reusable, elements. Compositional verification using assume/guarantee contracts, embodied by the Assume Guarantee REasoning Environment (AGREE) tool (reference 1), provides a functional property verification environment that is scalable to large, complex systems. AGREE translates the system architecture and contracts into a collection of model checking problems that are verified or falsified with counterexamples. AGREE can also check contract consistency and realizability (i.e. ensure that the contracts implementable).

This approach significantly reduces integration and verification efforts by identifying and correcting system design errors early in the lifecycle rather than waiting until system integration. Components in the system model are annotated with assume/guarantee contracts that include the requirements (guarantees) and environmental constraints (assumptions) that are defined as part of its development process. We then reason about the system-level behavior based on the interaction of the component contracts. By partitioning the verification effort into proofs about each subsystem within the architecture, the analysis will scale to handle large system designs.

In addition to expressing the functional properties, system safety and security properties should also be formally expressed as contracts and analyzed. Incorporating safety and security properties directly into the system models from which the system implementation is created helps to ensure accurate and consistent results. Reference 2 provides a description of an approach to model-based safety analysis.

**Architecture Is Correct**

The architecture model specifies the overall organization of the system. It defines interfaces for each subsystem and component, how they interact, and what data is shared. Properties to be verified can be divided into structural or behavioral properties.

We have chosen to model system architectures using the Architecture Analysis and Design Language (AADL). AADL has been developed to capture the important design concepts in real-time distributed embedded systems. AADL is a defined SAE standard (reference 3). AADL (see references 4 and 5 for an AADL introduction) is well suited to this domain, and provides an excellent mechanism for capturing the important details of system architecture and design. In AADL, the architectural model includes component interfaces, interconnections, and execution characteristics, but not their implementation. The language offers a high degree of flexibility in terms of architecture and component detail. This supports incremental development where the architecture is refined to increasing levels of detail and where components can be refined with additional details over time.

AADL describes the interactions between components and their arrangement in the system, but the components themselves are “black boxes.” The component implementations are described separately using model-based specification languages or traditional programming languages that are included by reference in the architecture model. This separation of implementation and architecture is an important factor in achieving scalability for the analysis tools. Separation of implementation from architecture also aids in the development and sustainment of software product lines by allowing for efficient upgrades, refactoring or even switching in 3rd party implementations.

Structural properties of the model can be checked statically. That is, they do not change over time. Properties are defined for AADL components to specify important configuration
data such as execution period, scheduling deadlines, worst-case execution time, bus latencies and data flow. This information can be used to support computation of CPU utilization, bus utilization, latency and schedulability analysis. Data in AADL models also provide the basis for generating configuration data for the kernel or operating system and synthesis of “glue code” that implements component interactions and access to kernel services.

Resolute (reference 6) is a tool for constructing an assurance case based on the structure of AADL architecture model. An assurance case provides the capability to address properties of the system that may be less precisely defined or that combine verification evidence from multiples sources. Assurance cases based on the architecture model may address structural or behavioral properties or both. Using Resolute further increases confidence that the system has been defined consistently and completely. As is the case the behavioral contacts expressed in AGREE, the assurance cases expressed in Resolute are attached to the system architecture. In so doing, system/product sustainment costs are lowered by allowing efficient impact assessment of component replacements or upgrades.

Components Are Correct

Once the system model has proven correct, it must be established that the components have been implemented correctly. This means that they must satisfy their requirements as specified in AADL contracts. Component implementations must also be verified to satisfy their contracts with the rest of the system. This can be done by model checking and/or by generating test cases from the contracts. Note that realizability checks ensure that the system model has defined components are “implementable”.

For example, we have developed automated tools for exporting AGREE contracts from AADL into the Simulink environment to support component verification by model checking. A similar approach can be used to automate verification of components implemented in other languages. In particular, languages, such as SPARK, that support formal verification should be considered.

System Execution Conforms to Architectural Model

The architecture model makes both explicit and implicit statements about how the system should execute. It explicitly specifies execution times and periods for tasks, binds threads and processes to CPUs, specifies connections between components, and routes messages on communication busses. As important are the implicit requirements contained within the architecture. For example, connections between components not explicitly defined in the model would mean that during execution there must be no data can flow between these components. This prevents unintended access to memory across component boundaries that might be intentionally exploited by an attacker or accidentally by a faulty component.

The operating system is responsible for carrying out or enforcing characteristics of the system as specified in the architecture model. The operating system is not explicitly modeled in AADL as a separate component or subsystem. Rather, it manifests itself in the specification of component execution properties, connections between components, and properties that define relationships between hardware and software in the system.

Therefore, an implementation requires the use of an operating system capable of enforcing these characteristics. Operating systems that enforce memory and time partitioning are widely used in avionics. These operating systems have primarily been verified using traditional software certification techniques (i.e. in accordance with DO-178C). Another approach would be to use an operating system kernel such as seL4 (reference 7). Both the explicit and implicit characteristics of the system are guaranteed by the seL4 kernel. seL4 was developed with formal proofs of correctness of its functional and security properties. These proofs extend from the requirements all the way down to the seL4 binary implementation.

System Implementation Corresponds to the Model

Addressing the four previous elements of architecture-driven assurance allows the system engineer to gain confidence in the quality of the architecture and the correctness of its specified behaviors. Nevertheless, unless the implementation matches the model, this confidence may not carry over into the actual implementation of the system. For this reason, it is important that the models be generative. This means that we can automatically generate the code and configuration data needed to build the system directly from the architecture and component models.

MODELING AND ANALYSIS TOOLS

Technology development programs have contributed to the creation of a number of inter-related but also separate tools and capabilities. The individual modeling and analysis tools are integrated into an engineering workflow. The individual tools and capabilities are introduced in the following subsections.

Requirements Development with SpeAR

High-level system objectives and capabilities are often expressed informally in forms such as use cases, concept-of-operations, SysML models, etc. Translating these high-level, informal representations to a formal language may present a challenge to even experienced system engineers. Our approach for handling this first step into formal representation is to use Specification and Analysis of Requirements (SpeAR) (see reference 8) to formally specify and rigorously analyze requirements.

With SpeAR, a user can formally capture requirements in an English-language-like syntax and perform formal analysis on those requirements. The user can prove formal properties
about the requirements, given the stated assumptions about
the environment or calling context. The expression and
analysis of requirements in SpeAR is independent of a
specific system architecture or design.

SpeAR provides a set of analyses to establish correctness,
completeness, and consistency of high-level requirement sets
prior to designing the system; preparing the way to begin the
process of expressing the system architecture and functional
properties.

Integrated Modeling Environment

The Rockwell Collins Integrated Modeling environment (see
Figure 3) is an open-source set of tools that provide system
architectural representations, architectural analysis and a
trusted build (architecture translation) capability.

![Figure 2, Open Source Integrated Modeling Environment](image)

The tool environment is based on the Open Source AADL
Tool Environment (OSATE). OSATE provides an interactive
textual and graphical environment for the development and
analysis of AADL models. OSATE is a mature tool that is
widely used for AADL-based tool development and
deployment. A large part of its success is based on its ability
to support extensions for new analyses and behaviors. This
makes our AADL technology readily available to the larger
AADL user community. See reference 9 for a more extensive
description of OSATE modeling, code generation and
analysis capabilities.

Rockwell Collins has developed several open-source OSATE
plug-ins and annexes. Plug-ins include architectural analysis
tools: AGREE, AGREE Simulator (SIM), Resolute and Test
Case Generation (TCG). Additionally, the Trusted Build tools
have been created to provide a means for the system
implementation to match the system models.

The plug-ins are updated/extended based on ongoing research
efforts. The updates continue to be available on an open-
source basis.

Architectural Analysis

**Proof - AGREE**

One of the barriers to formal verification of large systems is
the scalability of the analysis methods and tools. Rockwell
Collins developed the AGREE plugin for OSATE to
overcome this barrier. AGREE (reference 10) performs
compositional analysis allowing verification of system
requirements based on the composition of the component
contracts. By abstracting the implementation of subsystems
and software components as formal contracts, large systems
can be built up and verified hierarchically in an AADL model
without the need to perform a monolithic analysis of the entire
system.

Assume-guarantee contracts provide an appropriate
mechanism for capturing the information needed from other
modeling domains to reason about system-level properties. In
this methodology, guarantees correspond to the component
requirements. These guarantees are verified separately as part
of the component development process, either by analysis or
by testing. Assumptions correspond to the environmental
constraints used in verifying the component requirements. A
contract specifies precisely the information needed to reason
about the component’s interaction with other parts of the
system. Therefore, contracts can be used to develop portable
software components since a contract abstracts the component
implementation and specifies the interface and behavior that
the component must present to the rest of the system. The
implementation that fulfills the contracts can be variable; in
other words, any number of potential implementations can be
used so long as the contracts are fulfilled. Further, contracts
can be used to protect the intellectual property of separate
contractors since they permit system integration and
verification without exposing the implementations of
software components.

**Simulation - AGREE Simulator (SIM)**

The AGREE Simulator integrates with the AGREE analysis
tools to simulate AADL models with AGREE contracts.
While simulation is not sufficient to ensure a system design is
correct, it is a very useful capability that can provide quick
feedback on the basic functionality of a design during virtual
integration. A simulation environment with a user interface
that allows the user to provide inputs, step through execution
of the system, and set breakpoints and watch variables is
extremely useful for debugging both test cases and
counterexamples produced through formal verification. The
state of the simulation is displayed in both a graphical and a
tabular view. For the graphical view, the OSATE Graphical
Editor allows the user to annotate the graphical display with
simulator-specific information and to display the simulation
state on AADL graphical display.

**Assurance Cases - Resolute**

Resolute (see reference 4) is used generate and check
assurance cases associated with AADL models. An assurance
case is a methodology that describes claims about a system
and providing evidence supporting these claims. Assurance
cases are an excellent method to capture and document heterogeneous evidence related to system performance, function, security, and safety properties such as test results, analysis, and simulation to support the top-level claims. Previous assurance case tools have been completely separate from the actual system design artifacts. This made it difficult to keep the assurance case synchronized with the actual system design throughout its lifecycle. Resolute embeds the assurance case claims and rules in the architecture model itself, and permits system developers to make reference to and check structural properties in the AADL model, and to invoke other verification tools (e.g., AGREE) in order to construct an argument. Resolute uses the system modeled in AADL to generate an assurance case for the top-level claims and highlights where the assurance case fails if any of the supporting claims cannot be satisfied or have been invalidated by changes in the system design. Maintaining the representation of the assurance case claims and rules directly with the system architectural model ensures consistency with the design and enables lower lifecycle costs. Design trades and their impacts can quickly be evaluated.

Test Cases - Test Case Generator (TCG)

Automated generation of test cases from assume-guarantee contracts provides a rigorous, metrics-based approach to reducing verification efforts. Unlike compositional verification, that verifies a model for all possible combinations of inputs and reachable states, testing can only verify a tiny fraction of the possible behaviors of a system. Even so, it still plays a crucial role in verification of a system. Test cases generated for the overall system can be run through the simulator to discover discrepancies between the system level contracts and its component contracts before investing time and effort in compositional verification. Automatically generating tests for a legacy or vendor component provides a straightforward way for the supplier to test if the physical component actually satisfies its contract. Finally, it reduces the overall cost of verification by eliminating the cost of writing test cases by hand. Just as the assume-guarantee contracts allow a component’s required behavior and interactions to be expressed independent of its actual implementation, test cases generated from the same contracts allow a specific implementation to be tested against expected behaviors.

Architecture Translation - Trusted Build

For the design-time models and analyses to be meaningful throughout the system lifecycle, the system implementation must preserve the properties that have been established for the architecture and components. Trusted build is a tool that allows for the building of the binary for a system from its AADL architecture model. AADL model specifies the information needed to configure the operating system. It also specifies the models or source code needed to build the components in the system. Given this input, Trusted Build generates the files, configuration data, and small amounts of “glue code” needed to compile the system software. The transformations performed are simple enough that the resulting system can easily be determined to conform to the verified system models. The Trusted Build process and tools are described in more detail in reference 11.

ENGINEERING WORKFLOW

Incorporation of the concepts of architecture-driven assurance using modeling and analysis tools into the typical systems engineering workflow is shown in Figure 3. The work products resulting from the process are shown at the center of the “process V” and are primarily a set of architectural and analysis models used to ensure that the requirements and architecture are correct. The design and analysis activities (Requirements Analysis, System Architecture & Analysis, Subsystem Architecture & Analysis, and Component Design & Analysis) of the workflow would typically be highly iterative. As the architectural and behavioral analysis performed in these activities is virtual, iteration and refinement are efficient and effective.

![Figure 3, Architecture-Driven Assurance Engineering Workflow](image)

A high-level description of the process steps:

- Reference and objective architectures are used to guide, constrain and inform the system architecture and subsequent system design.
- For the purposes of this paper, the reference and objective architectures are shown as being captured in SysML and/or DoDAF views. The models contain information used in the Requirements Analysis and System Architecture and Analysis activities.
The Specification and Analysis of Requirements (SpeAR) integrated development environment (IDE) is used to derive high-level system requirements independent of a specific system architecture. The SpeAR IDE allows for formally specifying and rigorously analyzing requirements. In deriving the requirements to be analyzed, one might use information from the SysML models (e.g. use cases, sequence diagrams) or concepts of operations material.

The results of the formal analysis of requirements in SpeAR contributes to the system validation process—requirements are complete, consistent and correct.

System architectural models are developed using AADL and AGREE. AADL is used to represent structural elements, properties, and interconnections between elements. AGREE is used to represent behaviors (requirements) as assume-guarantee contracts. AGREE provides a set of analyses to ensure that requirements a complete, consistent and provably correct.

Test cases are automatically generated from the AGREE contracts. The test cases are used to be build test procedures for subsequent testing of the “as built” system.

Levels of the architectural hierarchy are further developed using AADL and AGREE, and are derived from the high-level system architectural models. Continued AADL and AGREE analyses are performed at this step.

Test cases are automatically generated from the AGREE contracts. The test cases are used to be build test procedures for subsequent testing of the “as built” subsystem(s).

Trusted build generates the software framework (glue code) directly from the architectural models. The glue code includes threading, partition scheduling, memory allocations, communication ports, etc. The implemented components are then inserted/integrated into the generated framework to form the system image.

CONCLUSIONS
The technology is available to enable transforming the system engineering workflow to better support defining, developing and verifying safety-critical avionic systems. The architecture-driven assurance method discussed in this paper incorporates the use of formal methods throughout the system engineering lifecycle. The methods and tools have been successfully demonstrated on recent programs such as DARPA High Assurance Cyber Military Systems (HACMS) (reference 11), NASA Compositional Verification of Flight Critical Systems (CVFCS) (reference 10) and Joint Multi-Role Technology Demonstrator (JMR TD) Architecture Implementation Process Demonstrations (AIPD).

The capabilities and tools continue to be enhanced and extended with ongoing technology programs focused on:

- Advancing the formal methods tools to allow systems engineers to design-in security throughout the development lifecycle
- Model-based safety analysis to enable efficient verification and validation of complex safety-critical systems
- Assurance of autonomous systems and behaviors

Architecture-driven assurance allows for applying a “virtual integration” methodology by creating analyzable models of the system architecture provides benefits such as:

- **Early detection and correction of design errors** – System modeling and automated analysis allow early detection and correction of design errors and identification of integration issues prior to implementing the system, which results in reduced cost and development times.
- **Validating implementations** – The use of formal specification languages for requirements reduces ambiguity, provides the ability to analyze and simulate requirements, and validate the implementation is correct.
- **Build what you analyze** – Automatically build software from architecture models; system implementation matches the models.
- **Development of product-lines** – Well-defined models of a system architecture provide a basis for identifying product-line opportunities and the specification of reuse strategies that can accommodate the use of 3rd party software, upgrades, design variants, and rapid prototyping of new systems. The hierarchical models separate the “what” from the “how” via component contracts, which allows for upfront analysis component replacement, or upgrade candidates.
- **System/product sustainment** – System architecture models facilitate the assessment of component replacements and/or technology insertions throughout the lifecycle of the system.
- **Model-based safety analysis** – Automated safety analysis based on the system architecture models to support meeting the system safety objectives (SAE ARP-4761 and SAE ARP-4754A).

Author contact: Dan Bliek dan.bliek@rockwellcollins.com

LIST OF ACRONYMS

AADL Architecture Analysis and Design Language
REFERENCES


9. OSATE. www.osate.org
