Joint Common Architecture (JCA) Demonstration Architecture Centric Virtual Integration Process (ACVIP) Shadow Effort

Alex Boydston
Electronics Engineer
US Army ADD/JMR
Redstone Arsenal, AL

Dr. Peter Feiler
Principal Researcher
AADL Tech Lead
CMU SEI
Pittsburgh, PA

Dr. Steve Vestal
Distinguished Scientist
Adventium Labs, Inc.
Minneapolis, MN

Bruce Lewis
AADL Chair
US Army SED
Redstone Arsenal, AL

ABSTRACT

Challenging problems associated with system software complexity growth are threatening industry's ability to build next generation safety critical embedded systems including helicopter avionics systems. Contributors to these problems include the growth of software, system integration, and interaction complexity exacerbated by ambiguous, missing, incomplete, and inconsistent requirements. Problems continue to hamper systems in the areas of resource utilization, timing, safety and security. A new approach called Architecture Centric Virtual Integration Process (ACVIP) which is based on the Society of Automotive Engineers (SAE) Standard AS5506A Architecture Analysis and Design Language (AADL) is being developed and investigated by the US Army to address these challenges. ACVIP is a quantitative, architecture-centric, model-based approach enabling virtual integration analysis in the early phases and throughout the lifecycle to detect and remove defects that currently are not found until software and systems integration and acceptance testing. In an effort to investigate such an approach, the Government, in conjunction with researchers from Carnegie Mellon University (CMU) Software Engineering Institute® (SEI) and Adventium Labs®, are conducting ACVIP requirements, safety, and timing analyses in parallel with the Joint Common Architecture (JCA) Demonstration (Demo).

INTRODUCTION

The United States Army Aviation Development Directorate (ADD) and Software Engineering Directorate (SED), teamed with Carnegie-Mellon University (CMU) Software Engineering Institute® (SEI) and Adventium Labs®, are currently conducting Science and Technology (S&T) research on the Joint Multi Role (JMR) Technology Demonstrator effort's Joint Common Architecture Demonstration (JCA Demo) Project (Ref. 1) to investigate and mature a concept called Architecture Centric Virtual Integration Process (ACVIP). ACVIP is a DoD process fashioned after the aviation research study called System Architecture Virtual Integration (SAVI) (Ref. 2) performed by a consortium of aerospace organizations led by Aerospace Vehicle Systems Institute (AVSI). Like SAVI, the purpose of the ACVIP is to address the affordability and associated risks of developing complex software intensive systems through early virtual integration and analysis before implementation. The JCA Demo provides a first look for the Government to gain experience with several analyses and tools to determine if a subset of ACVIP analyses detect any integration issues or software defects prior to or during their manifestation in the JCA Demo effort. ACVIP analyses were conducted as a shadow effort during the demo and did not constitute the execution of an ACVIP review process.

The JCA Demo was designed to exercise the Future Airborne Capability Environment (FACE™) (Ref. 3) Technical Standard and Tools and Joint Common Architecture (JCA) by producing and integrating a standard conformant Data Correlation & Fusion Manager (DCFM) software component into an unidentified system (i.e., later revealed as the Modular Integrated Survivability (MIS) system). The ACVIP Shadow effort involves the modeling of the MIS system architecture, the DCFM component and their associated requirements using the Architecture Analysis and Design Language (AADL) (Ref. 4) and Open Source AADL Tool Environment version 2 (OSATE2) (Ref. 5). The analyses are similar to, but more advanced than, what has been demonstrated during the SAVI program. SAVI conducted a return on investment (ROI) study citing that for a new aviation system with the complexity of 27 million software lines of code (SLOC),an estimated nominal savings of about $2.4B out of $9.2B, i.e., about 25%, could be realized from using a systems architecture virtual integration process to reduce software rework (Ref. 6). This represents the complexity level of advanced aircraft in 2010 which suffered significant software system integration issues.

This paper will address the following:

1) JCA Demo Background

2) Overview of the AADL and OSATE2

Presented at the AHS 71st Annual Forum, Virginia Beach, Virginia, May 5–7, 2015.

3) ACVIP in the context of lifecycle acquisition

4) Discussion of the ACVIP Requirements, Safety and Timing Analyses Methods along with JCA Demo Experience with each

5) Findings and lessons learned from the JCA Demo ACVIP Shadow effort.

6) Projection of future maturation and plans for ACVIP beyond the shadow effort

JCA DEMO BACKGROUND

In February 2013, the JCA Demo Broad Agency Announcement (BAA) was released on FedBizOps.gov without the incorporation of the ACVIP Shadow task. The JCA Demo is a Technology Investment Agreement (TIA) requiring the delivery of a Data Correlation and Fusion Manager (DCFM) software component built to minimal textual requirements and a data model that contained information per the FACE Standard. The DCFM was to be integrated into the Modular Integrated Survivability (MIS) system with the MIS team acting as the system integrator. TIAs were awarded to two separate DCFM vendors, Honeywell Aerospace® and Sikorsky®-Boeing® Teams.

The US Army's Aviation and Missile Research Development and Engineering Center (AMRDEC) Aviation Development Directorate (ADD) and Software Engineering Directorate (SED) contracted with CMU SEI and Adventium Labs after the JCA Demo BAA was released to conduct shadow analyses with the AADL related to ACVIP. The focus of the JCA Demo ACVIP Shadow effort was to conduct requirements, safety and timing analyses to identify issues in each of these areas prior to DCFM integrations into the MIS architecture.

The JCA Demo project schedule and funding limitations prevented implementing a full ACVIP process. Two AADL models were created, a conceptual architecture model for requirements and safety analyses, and a runtime architecture model reflecting the timing of the intended operational system. Normally the ACVIP process would refine a single conceptual model to create the runtime model and analyses would be generated from the same model incrementally. If ACVIP were fully implemented the JCA Demo analysis would have begun before solicitation release and continued throughout employing a single source of truth architectural model.

For JCA Demo, communication between the DCFM developers, the system integrator and the ACVIP analysts was tightly controlled. The purpose of this segregation was to see if the ACVIP analysis would uncover issues that would later manifest during the course of the demonstration. It was important for ACVIP to find issues without listening into discussions between the component developers and system integrators. It was equally important that the DCFM developers and integrator were not tipped off to issues discovered by ACVIP.

The AADL modeling of requirements discovered significant problems with minimal effort prior to award of the TIAs. For example, a problem related to response time requirements was discovered through early end-to-end latency analysis of the functional architecture model targeted to an ARINC653 platform. Since this was a shadow, the contractors developing the DCFM had to discover and clarify these issues to proceed with the development. From this experiment it was evident that pre-solicitation requirements analysis would have greatly benefited the demonstration in reducing requirements gaps and errors.

AADL/OSATE OVERVIEW

The AADL is an Architecture Description Language (ADL) (Ref. 7) designed from its beginnings in Defense Agency Research Programs Agency (DARPA) studies. Its purpose is to enable analysis across integrated components predicting qualities of the integrated system and then the generation of the system in accordance with the models and analyses. AADL started life as the MetaH language at Honeywell in the early 1990's on the first research programs devoted to developing ADLs. MetaH was intended for use in the domain of embedded real-time systems where predictable correctness is required supporting high assurance and safety critical systems. The Government appointed Mr. Bruce Lewis (US Army SED and one of the authors of this paper) as the lead for this project after it was started by the US Navy. The concepts were refined over three DARPA programs, but from the beginning, the language stressed component-based, analysis-driven, architecture centric development with automated compliant implementation, as developed by the Principle Investigator, Dr. Steve Vestal, also an author of this paper. In Army internal projects, the concepts and capabilities proved so valuable that the language was taken to the SAE to create an international standard (i.e., SAE Aerospace Standard 5506) and to enhance the language to meet additional research and industrial requirements. Dr. Peter Feiler, also an author of this paper, became the Lead Architect for this SAE language standard, known as AADL. AADL incorporates an extended set of MetaH concepts with concepts from the DARPA funded CMU architecture interchange notation, ACME, to achieve extensibility to accommodate multiple analysis dimensions from the same model.

The AADL was specifically designed to support incremental refinement and analysis of a system from its conceptual architecture to its implementation runtime architecture. AADL is a strongly typed language with well-defined semantics. Strong typing provides consistency within the model, e.g., ensures that only components of the appropriate type are connected. Well-defined semantics ensures analysis tools interpret the model the same way and produce consistent results. For example, the execution

behavior of tasks is defined in the standard with a hybrid automata specification that allows for formal analysis using temporal logic. As a result, AADL achieves portability across model editing tools and integration across contractors into a single specification captured in AADL. The extensibility mechanisms of AADL, annexes and property sets, ensure that this model consistency is maintained. These extensions allow new domains of analysis to be supported by annotating the core language that is converted into representations for specific analysis tools, e.g., into timing models or fault tree representations.

Because the AADL is an architecture language intended to support each phase of system development, it was designed to explicitly support incremental refinement and analysis throughout the system lifecycle. Early analysis can be run on conceptual architecture models that may be incomplete. These models can still be checked for consistent integration. Any aspect of the architecture can be refined to the level required for that subsystem's level of development.

The AADL directly supports modeling and analysis of functional architectures as well as component, task and communication architectures of software systems, their deployment onto distributed hardware platforms, and their interfaces with the physical system. The functions are mapped to components during the development process. These aspects are necessary to analytically predict operational quality attributes, i.e., the effects of software running on hardware to control physical system elements, to ensure they meet non-functional requirements.

To find the system integration issues virtually, before physical integration, we need not only early analysis, but also analysis at each stage of development, at higher and higher levels of fidelity, to discover and correct issues through analysis and verification. The actual system implementation can then be integrated in accord with the verified model. This is critical for trusted integration, a research objective of AADL and ACVIP. The system can be integrated before the components have been populated with code, providing a very early prototype of the architecture structure and behavior. Then as code is developed, via code generators for components (like Mathworks® MATLAB/Simulink™ and Esterel® SCADE™), the architecture can be populated and re-integrated for integration testing and refinement of the predictive models.

The AADL component categories include software components (e.g., processes, threads (tasks), subprograms, and data, as well as thread groups and subprogram groups), hardware components (e.g., processors, buses, memory, devices, as well as virtual processors (e.g., partitions and multi-core hypervisors) and virtual buses (e.g., protocol layered buses). AADL also has specific semantics for types of ports and connections. At the next level up, the AADL has system components that encapsulate hardware and software components, into subsystems at multiple levels up to the system itself. The AADL also supports abstract components which can be specified before a decision is made relative to the component category. The abstract and system components provide a level for conversion from System Modeling Language (Ref. 8) (SysML, i.e., an extension of UML used for systems engineering) to AADL. Then the architecture can be extended related to its real-time attributes and its component categories in the AADL in a standard way designed for analysis and implementation.

AADL components have both an external view (Type) and an internal view (Implementation). Both can be extended and refined to create new components, reuse existing components and specify the components more completely, incrementally refining until the final component is fully specified. Properties and annexes can be associated with each component category, as allowed in the language. These properties provide information about the component related to many aspects and domains of analysis. The AADL ARINC653 Annex (Ref. 9) can be used to model ARINC653 (Ref. 10) partitioned architectures and properties; the AADL Behavior Annex (Ref. 9) can be used to model the internal behavior of the component; and the AADL Data Modeling Annex (Ref. 9) can be used to define the data types to support code generation. Properties can be associated related to safety, security, timing, binding to hardware, etc. AADL Standard annexes and properties provide a very rich set of capabilities for analysis. More recently, AADL has added functional components to support requirements and hazard analysis as well as refinement into the standard software, hardware, and system components for analysis and system building.

The AADL language very naturally supports the ACVIP process. The SAVI international industrial (aviation integrators and suppliers) and Government (DoD, NASA, FAA) consortium determined that the key problem driving very high costs and high risks in aviation systems was the need to be able to keep the system integrated throughout the development process. Their mottos, "Integrate, Analyze, then Build" and "Keep the system integrated throughout the development process", lead to a virtual integration process throughout development, as well as keeping models consistent as development proceeds. SAVI selected the AADL after reviewing all ADL's for this purpose, especially related to the software-reliant part of the system. The ACVIP is a subset of the SAVI process we can build on today. Both SAVI and ACVIP are centralized on virtual integration, conducted incrementally, across suppliers and the system integrator, covering multiple domains of system analysis. The AADL directly supports integration analysis against a single truth model (integrating the most recent data) for software reliant systems, incrementally, predictably, across critical analysis domains (such as safety, security, timing, scheduling, latency, utilization, etc) for aviation (and medical/nuclear/automotive, etc) systems. It then supports advanced approaches to system implementation. ACVIP and SAVI support contractor freedom to select their languages, analysis methods, tools

and models but then integrate the results into the integrated model expressed in the standard semantics of AADL and other languages for incremental integration analysis.

AADL specifications can be processed in the OSATE2 toolset which provides both a textual and graphical user interface for editing. OSATE2 is an open source freeware product based on Eclipse that provides the reference implementation of AADL. OSATE2 also integrates multiple analysis tools that can be used within the toolset. OSATE2 can export data to other analysis tools and has integrated capability for code generation, just recently demonstrated for ARINC653. In the near future, OSATE will also support a user friendly form based input to assist engineers serving different roles

The AADL Inspector (Ref. 11) is another toolset that can be used to develop AADL specification with a number of analysis methods, most of which have been developed in Europe. It is a commercial toolset and includes simulation capability. STOOD is an established commercial toolset that supports development in HOOD and AADL (Ref. 12).

Another AADL toolset has been developed in Russia by the Russian Academy of Science in partnership with the GosNIIAS (Russian State Research Institute of Aviation Systems) aviation systems lab for Integrated Modular Avionics (IMA) architectures (Ref. 13). This toolset has been partially released for public use, providing the AADL graphical and textual editor, but not releasing advanced analysis methods for IMA development and implementation.

Each of these toolsets can exchange AADL models for selecting specific analysis tools of interest supported. So by chaining tools, the developer can leverage strengths of each.

There are also a number of other AADL tools, like the TASTE, COMPASS, and the D-MILS toolsets (Ref. 14). COMPASS and D-MILS extended the AADL language and are limited to European Union (EU) use. TASTE developed a "zero coding" approach to satellite system development and upgrades using a system engineering level interface, transformation into AADL, domain specific component code generators, analysis, and then automated AADL integration and generation of complete load images for the system.

ACVIP IN THE CONTEXT OF ACQUISITION

DOD 5000.02 Instruction for the "Operation of the Defense Acquisition System" (Ref. 15) addresses the acquisition process as shown in Figure 1. From the earliest stages of acquisition requirements are formulated from stakeholders in the Initial Capabilities Document (ICD) and Capability Development Document (CDD) and derived to system, hardware and software requirements. It is important that the validity and completeness of these requirements be checked early, for it is through the derivation of these requirements that mistakes are made which result in increasing cost of systems that can go all the way through to the Operations & Support phase.

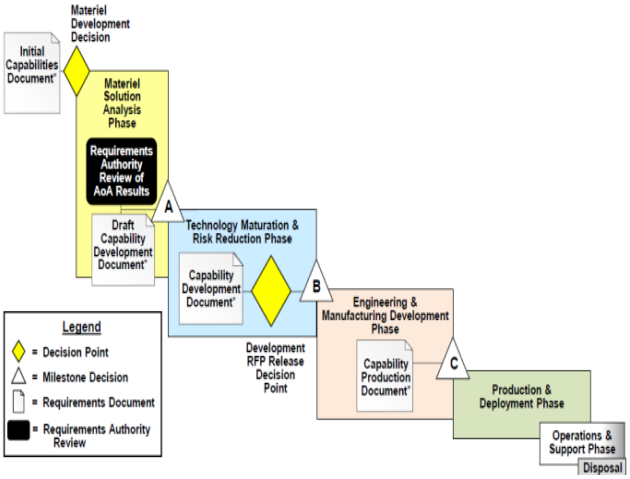

Figure 1: DoD 5000.02 Acquisition Lifecycle (Ref. 15)

Embedded within this lifecycle are standard reviews such as System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), Test Requirements Review (TRR), and Production Readiness Review (PRR). In a model based ACVIP approach, it is envisioned that as the requirements lead to conceptual architecture(s) that virtual analyses, trades and verification can be conducted on the system as early as the Technology Maturation & Risk Reduction Phase and perhaps earlier. Stakeholder and system requirements documents often contain an implied architecture. Such conceptual architectures could be analyzed virtually to mitigate requirement issues. There have been numerous examples of acquisition programs where requirements analysis has been deficient resulting in inadequate resources (memory, storage, processing bus bandwidth, and ample timing boundaries), residual safety risks, deficient security, and incomplete qualification. Oftentimes, the errors that exist within the requirements result in a system full of risks and too expensive to fix, and ultimately driving to a Nunn-McCurdy program breech.

The DoD 5000.02 instruction states, "… the Program Manager will integrate modeling and simulation activities into program planning and engineering efforts. These activities will support consistent analyses and decisions throughout the program's life-cycle. Models, data, and artifacts will be integrated, managed, and controlled to ensure that the products maintain consistency with the system and external program dependencies, provide a comprehensive view of the program, and increase efficiency and confidence throughout the program's life-cycle." It is envisioned by SAVI and ACVIP that these problems can be averted through early virtual integration and analysis via an architectural centric model based approach.

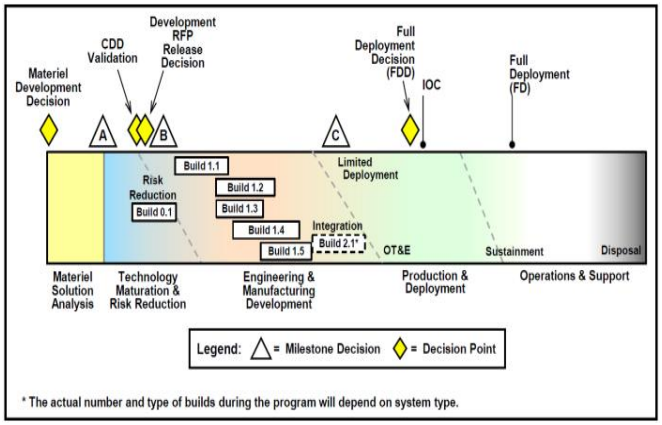

Figure 2: Defense Unique Software Intensive Program (Ref. 15)

The acquisition approach for software intensive mission systems, as shown in Figure 2, can and should be augmented through solicitation via a specification model. Prior to the solicitation, high level analysis of requirements, timing, resources, safety and security can be conducted using the specification model. After release, the responders to the solicitation can utilize the specification model to create potential early system solutions in a very preliminary design model that can be used by the Government to conduct more refined analyses and trade studies to determine the best approach(s) to meet the requirement. The system integrator can continue to communicate the model specification to its component suppliers to obtain their respective component models. These component models will act as the component specifications and interface descriptions and allow the integrators to perform virtual integration and analysis. Once selection is made by the Government the winning solution can be even further refined and analyzed. As the model is matured it can be evaluated and analyzed at different program phases in an increasingly hierarchical manner to identify issues for correction before anything is actually built, coded or integrated. The architectural model(s) would be contained in a model repository remaining integrated, up-to-date and under configuration management to be available to multiple engineering disciplines that could rely on this as the single source of truth.

Challenges exist with this vision that must be addressed. This includes determining the appropriate time in the acquisition lifecycle to apply ACVIP (e.g., Material Solution Phase, Technology Maturation & Risk Reduction Phase, Engineering & Manufacturing Development Phase, etc.). Also, translation and exchange of models among different languages (e.g., UML, SysML, AADL, MatLab/Simulink and SCADE) and tools needs to be worked to allow government, integrators, and component suppliers to communicate seamlessly. Business issues like protection of intellectual property and the formulation of new profit models must be overcome as well. Lastly, and most importantly, the tools must be matured to a Technology Readiness Level (TRL) to enable users to adapt and use the analysis processes and tools effectively.

JCA Demo was a first demonstration for Army Aviation to acquire software using a model to communicate most of the requirements. In addition, AADL was used post-BAA release to analyze the requirement documents, and to perform safety and timing analyses on a model of the virtually integrated component within the MIS system to discover potential issues before actual system integration. JCA Demo provided the first step for the Army toward maturing the tools for ACVIP.

ACVIP ANALYSES

Architecture Capture Guidelines for ACVIP

As part of the JCA Demo effort, the ACVIP research team is documenting guidelines to help engineers develop and analyze AADL models in support of an architecture centric virtual integration process. ACVIP applies across development phases, starting with requirements engineering and going through verification and qualification. Different kinds of information at different levels of detail are used in the different phases. The AADL ACVIP modeling guidelines support this by identifying four general levels of abstraction for AADL models:

- Functional architectures capture functional requirements but with little or no information about how those functions will be encapsulated in components.
- Conceptual architectures specify how a system is decomposed into software and hardware components and the interfaces between them. Conceptual architectures are used during architecture trade studies and acquisition planning.
- Design architectures specify detailed performance characteristics of individual components, including internal design detail to the level required to support the analyses desired.
- Implementation architectures specify details needed to integrate and verify an overall system; for example, data that can be used to automatically generate configuration files or perform model-based testing.

The guidelines provide advice to technical project management and engineers as they make decisions about milestones at which models are developed and exchanged, the level of detail to be captured, the analyses to be carried out at each milestone, ways to capture information in AADL and methods for analysis. The guidelines also discuss some supporting processes: configuration management and model exchange, trade space exploration and architecture optimization, and liaison with airworthiness and security approval authorities.

Architecture Led Requirements Specification (ALRS) Analysis Overview

Current requirement engineering practice results in textual stakeholder and system requirement documents. Studies show that ambiguous, missing, incomplete, and inconsistent requirements lead to sizeable effort in clarifying them. Often the system boundary is not clearly specified and different requirement statements may refer to a system or one of its subsystems. The objective is to turn a system requirement specification into a contract that a system implementation must meet. This is then demonstrated through virtual validation and verification.

ACVIP addresses requirement capture and specification by an Architecture Led Requirements Specification (ALRS) analysis process. This process is currently being matured with tool support and leverages the AADL Requirements Definition and Analysis Language (RDAL). The requirements analysis addresses requirement quality characteristics of IEEE 830-1998 (Ref. 16) and adapts the eleven step process outlined in the Federal Aviation Administration (FAA) Requirements Engineering Management Handbook (Ref. 17). The process leverages the representation of the system and its operation environment as an AADL model. For that purpose we adapt the CPRET (Ref.18) representation of a system defined by the *Association Française d'Ingénierie Système* which is shown graphically in Figure 3.

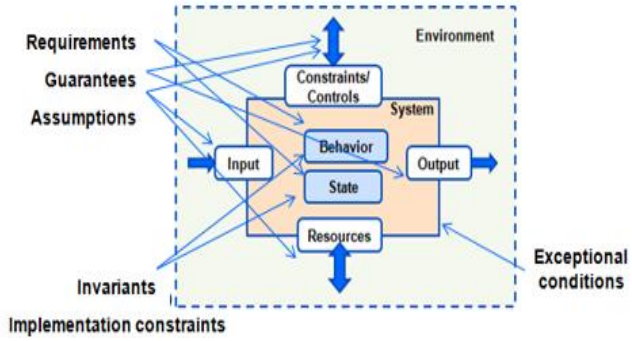

Figure 3- Elements of a System Specification

In the ALRS analysis process a user maps the information found in existing requirement documents to elements of an AADL model of the system. This model has captured the interactions of the system with entities in the operational environment with requirement specifications. This process clarifies whether the requirement is for the system or one of its subsystems, quickly identifying use of multiple terms for the same entity, and ambiguous or conflicting requirement statements. Such a mapping of requirement statements into the model also lets the user quickly see whether requirements have been specified for all interaction points with entities in the operational environment.

In a next phase the user utilizes utility trees that are the output of a Quality Attribute Workshop (QAW) (Ref. 19) or an Architecture Tradeoff Analysis Method (ATAM) (Ref. 20). They take non-functional properties, also known as operational quality attributes, and turn them into a concrete requirements specification that can be measured and verified. Prioritization of the utility tree leafs driven by mission goals help the user ensure that critical requirements are well-specified. Such a utility tree is shown in Figure 4.

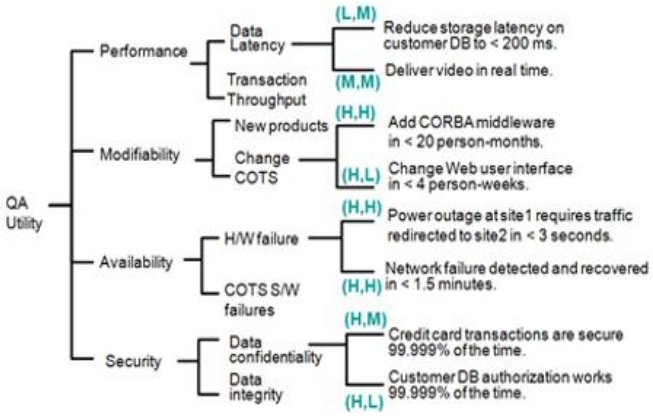

Figure 4: Quality Attribute Utility Tree

A third phase addresses exceptional conditions that may be encountered during system operation. These exceptional conditions impact safety, reliability, and security of a system. An analysis from a safety perspective is elaborated on later in a discussion on safety. Note that since we have a model-based representation of the system specification, the user can utilize an ACVIP workbench, such as the OSATE2 tool environment for AADL, to check for inconsistencies in the specification, e.g., check if the expected inputs and outputs match. The user can also perform quantitative analysis of the model. For example, flow latency analysis can be used to determine whether response time requirements are achievable, whether budgets for physical resources, such as electrical power and mass, or computer resources are realistic and result in sufficient margins for uncertainty and desired spare capacity.

Requirements Analysis Process and Results on JCA Demo

Prior to the JCA Demo DCFM awards, the ACVIP team from CMU SEI conducted requirements analyses based on the requirements and data model provided in the JCA Demo BAA and the MIS Stakeholder and Systems Requirements documents. This analysis identified shortcomings in the system-level and component-level requirements, some of which were also identified by the DCFM developers. Following the provision of initial derived DCFM developers' requirements, further requirements analysis was conducted to elicit additional integration issues. The ACVIP shadow effort for the JCA Demo performed analyses using AADL models of the DCFM integrated into the MIS system. In the process the team discovered inconsistencies, and

missing requirement information in the original documents, as well as defects related to safety, latency, and timing / resource utilization.

In the process of performing this mapping a partial AADL model of the conceptual and functional architecture were developed including both the "architecture" of the system in its operational context, and the system in terms of its subsystems as far as they had been reflected in the original requirements documents and UML model. This model clarified issues of system and subsystem boundaries. The resultant architecture model was generalized into an aircraft survivability situational awareness (ASSA) system, creating a reusable reference architecture for the domain of use. This ASSA system incorporates the MIS and the DCFM, both of which provide several functional services. This is illustrated in Figure 5 with three services for MIS. Two services are infrastructure services that are provided in a layer below the situational awareness system, i.e., the data conversion service, and the data management service. The third service, a health monitor, resides in a layer above the situational awareness system to detect and report any exceptional conditions in the operation.

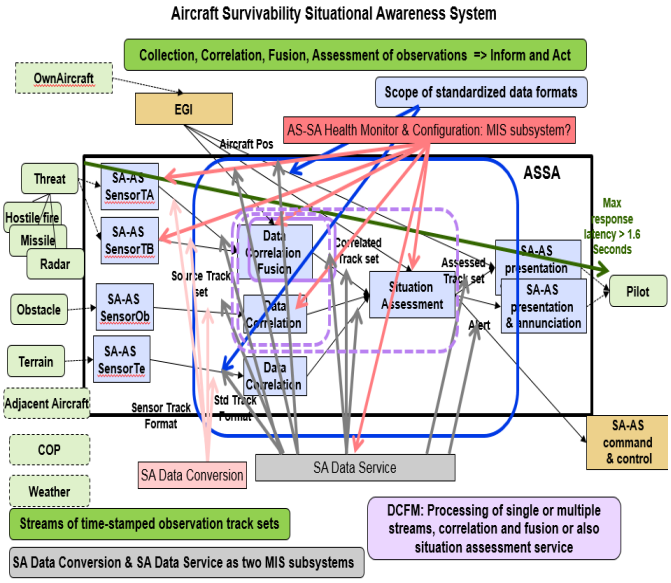

Figure 5 – Layered Architecture of ASSA System

The resultant annotated AADL model of the ASSA System clearly identifies how much of the system architecture has been prescribed by the requirement specification. This awareness helps clarify whether this was intentional, or whether requirements should be rephrased to become requirements of the enclosing system, leaving design choices to the developer.

The resultant functional architecture also became the basis for quantitative analysis of the ASSA early in development, e.g., pre-PDR. As Figure 5 shows, the model included end- to-end flow specifications of a critical flow to represent response time requirements. A UML sequence diagram from the original documentation was modeled as an analyzable interaction protocol across ARINC653 partitions. The latency analysis capability of OSATE2 informed us of the latency overhead contributed by this protocol, and its effect on the critical flow, i.e., that in the best circumstances the requirement can barely be met.

Architecture Led Safety Analysis (ALSA) Overview

The CMU SEI also conducted a safety analysis of the ASSA using an Architecture Led Safety Analysis (ALSA) process as part of the JCA Demo shadow effort. The user annotates an AADL model with fault information utilizing an error propagation ontology as illustrated graphically in Figure 6. The error propagation ontology addresses issues of service omission, commission, value, timing, rate, sequence, replication, concurrency, authorization, and authentication errors. The propagation paths between system components are derived from the architecture specification itself.

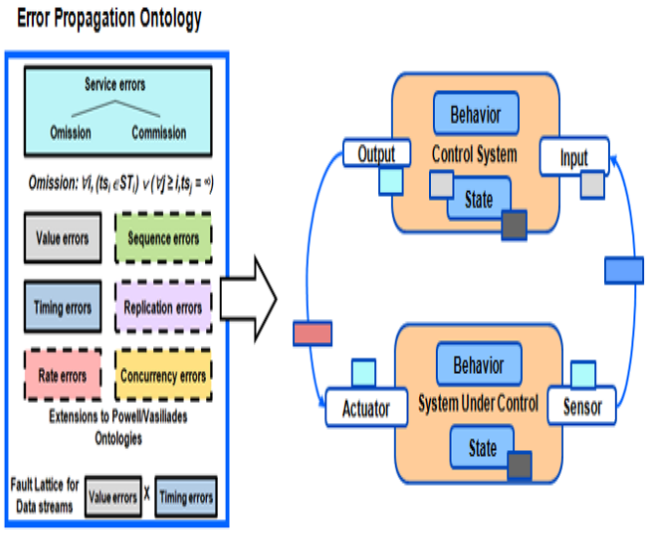

Figure 6- Identification of Hazard Sources and Impact

This process leverages method and tool support through AADL and the AADL Error Model Version 2 (EMV2) Annex (Ref. 21) to support SAE ARP-4761 (Ref. 22) best system safety analysis practices, such as an FHA, FMEA and FTA. The analysis models, such as a fault tree, are generated from the annotated AADL model, and then processed by a FTA tool. In the case of FHA and FMEA the respective reports are generated directly from the annotated AADL model – as shown in Figure 7.

Safety Analysis Process and Results on JCA Demo

System safety analysis guidance, such as SAE ARP-4761, recommends that the user perform a functional hazard assessment (FHA), a failure mode and effect analysis (FMEA), and a fault tree analysis (FTA). The FHA focuses on hazards that may lead to catastrophic events. As a result of these analyses, design assurance levels (DALs) are assigned to different subsystem hardware and software in an

aircraft. While the JCA Demo BAA set the DAL to level E for the DCFM, this ACVIP Safety Analysis exercise used level C for the situational awareness system for aircraft survivability.

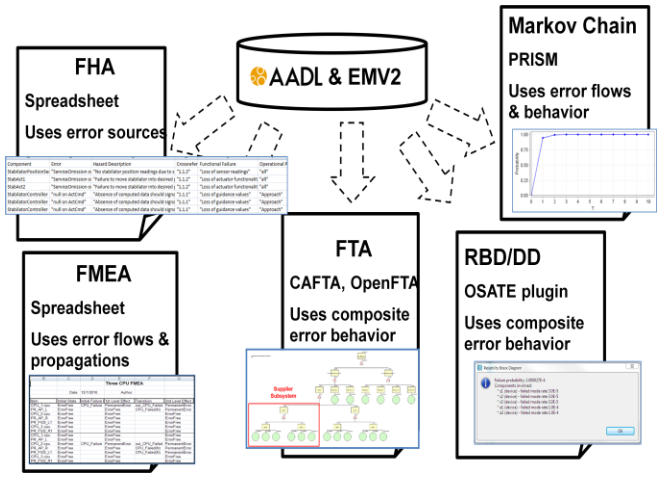

Figure 7- Safety Analyses from Annotated AADL Models

In the SAVI initiative the SEI recently demonstrated how the SAE ARP-4761 process can be supported by an AADL model annotated with fault information using the Error Model Annex standard for AADL on an aircraft wheel braking system. FHA, FMEA, and FTA reports as well reliability/availability analysis reports have been generated from safety analysis performed with such a model.

An Architecture-Led Safety Analysis (ALSA) process was conducted for the JCA Demo ACVIP shadow project. For that purpose the CMU ACVIP researchers started with the hazards presented to the pilot by the ASSA. In addition to the hazard of complete failure of providing the ASSA service, the hazards considered included providing false information such as false positives in the form of alerting the pilot of threats and obstacles that do not exist, false negatives such as not alerting the pilot when these threats and obstacles exist. In addition the timeliness of information was taken into account, i.e., how much information delay is acceptable to the pilot. Subsequent to citing the hazards, the potential error sources were systematically identified that can propagate as one of the identified hazard categories to the pilot. A fault ontology provided as part of the AADL Standard Error Model annex was used as a checklist of fault propagation categories to consider in the process.

The insights from this analysis lead to a set of derived safety requirements for the health monitoring system that were lacking in the original System Requirement document. These requirements were captured in the annotated AADL model of ASSA. The primary focus of the health monitoring system was on detection and reporting, i.e. it is responsible for recognizing when one of the identified hazard conditions occurs and then informs the pilot to that effect. A second set of requirements focused on minimizing the impact of the different fault contributors, i.e., to express fault isolation tactics as a set of derived safety requirements. A third set of requirements addressed the ability to recover back into a normal operational state. In other words, the resulting requirement specification provided a clearer indication of expected functionality.

A hazard analysis of this form not only examined failure of individual components, but also whether the interaction between components could lead to a hazard contributor. An example of such a contributor in the ASSA is the fact that the interaction between sensors providing new data to MIS and DCFM requesting data concurrently could potentially lead to concurrency issues which result in corrupted data, which in turn can result in false positives or false negatives.

Architecture Led Timing Analysis Methodology and Tools

For distributed heterogeneous computer systems, specifying and analyzing end-to-end timing requirements that result in satisfactory mission performance of the overall vehicle remains a challenging multi-disciplinary problem that involves the physical sciences and human factors as well as computer science and engineering. Different requirements models and allocation and scheduling methods are used for different functions and equipment. For example, networks typically use a different scheduling method than processors. Feedback control software uses a periodic sampled data design pattern, while message handling software often uses an event driven queued data design pattern. Today there is no single method or tool that can analyze all of them. Two broad approaches to timing analyses are simulation (testing executable models) and schedulability analysis (applying math to bound values). The two have strengths and weaknesses and can complement each other. In this project, schedulability analysis was the focus. In a survey the ACVIP timing analysis team identified sixteen available schedulability analysis tools, each suited for different scheduling algorithms and software applications and computing equipment.

The system architect must select a set of development methods and tools that are suitable for the subsystems and components selected for the mission system. The selected development tools need to be integrated, just as the components of the mission system must be integrated. The ACVIP team created and used a compositional timing analysis framework during the JCA Demo that allowed us to select a set of analysis tools suited for the different kinds of subsystems in the mission system. The framework compositionally applies the tools so that dependencies between subsystems are taken into account when producing an overall end-to-end timing analysis.

The timing analysis framework developed by Adventium and applied to the JCA Demo system translates different parts of an AADL model into the native input

formats of selected back-end schedulability analysis tools. There are dependencies between these multiple tool-specific models, e.g. a task set hosted on a processor analyzed by one tool may send messages over a network that is analyzed by another tool. The framework extracts analysis results from some tools when generating input models for others. Because there may be cyclic dependencies between different parts of the system model, analysis must be performed iteratively until global convergence is achieved. The final analysis results from the different tools must be combined to check end-to-end timing requirements specified in the AADL system model (Simon Kunzli, 2007; Rob Edman, 2015).

The Modeling and Analysis Suite for Real-Time Systems (MAST) (Ref. 23) and the Separation Platform for Integrating Complex Avionics (SPICA) (Ref. 24) were selected as the initial tools to integrate into the timing analysis framework as shown in Figure 8 (the Framework for Analysis of Schedulability, Timing and Resources, FASTAR (Ref. 25)). These selections were made because MAST can analyze switched Ethernet networks and SPICA can analyze ARINC 653 style schedules. An Ethernet network and ARINC 653 partitioning were used in the JCA Demo.

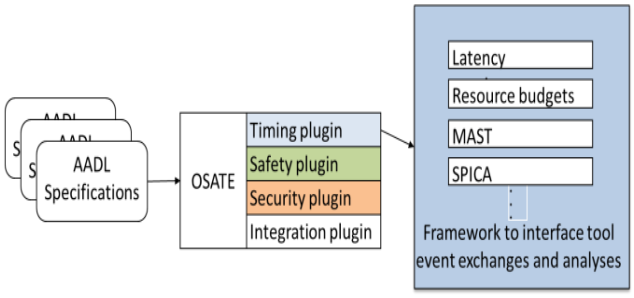

Figure 8- Timing Analysis Tools Framework Based on AADL

Although most of the workload in our demonstration system was hosted on an Ethernet and an ARINC 653 compute module, there were several pieces of sensor equipment (some simulated, some actual) and a display subsystem for which the ACVIP timing analysts had no internal design information (and insufficient project resources to model them even if the ACVIP timing analysts did have the internal design information). This is expected to be a common situation; therefore, the analysis framework allows "black box" modeling of subsystems. This allows the developer to enter interface timing properties for these subsystems into the model (e.g. message send and receive rates and latencies through the subsystem). The analyzer assumes "black box" subsystems will comply with their specified interface timing properties when doing end-to-end analysis and verification.

Timing Analysis Process and Results on JCA Demo

For the JCA Demo, a decision was made to perform timing analysis on a design architecture model. This decision was made both to gain experience with multiple development phases and modeling guidelines and to stress-test the timing analysis framework.

Both conceptual and design AADL models were developed. The conceptual architecture model was primarily based on a Microsoft® Word™ document that described the overall JCA Demo system architecture, a Microsoft Word document that described the derived interface requirements for a major subsystem, a Microsoft Word document and a data model included as part of the solicitation for the DCFM software component, and the JCA Demo system architecture configuration contained in the UML. Although conceptual models were not subjected to schedulability analysis (other types of timing and resource analyses are more appropriate at the conceptual architecture phase), this allowed the ACVIP researchers to exercise more of the modeling guidelines, including guidelines for capturing traceability between models at different abstraction levels using AADL language features to extend and refine component models with increasing amounts of detail.

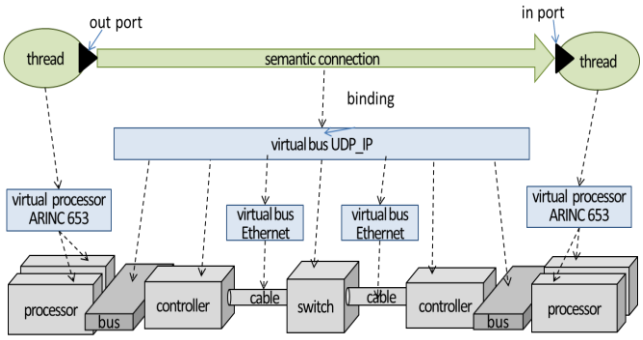

Figure 9 Architectures are Layered

As the ACVIP timing analysts built the AADL models for the JCA Demo system, they uncovered more detailed requirements for methods and tools needed to support architecture modeling and analysis. First, architectures are layered as illustrated in Figure 9. Layers introduce infrastructure software and affect timing properties such as system overheads. AADL allows virtual resources and layers to be modeled, but guidelines are needed and tools need to support those guidelines. Second, architectures have different clock synchronization domains. For example, the JCA Demo system hosted portable interoperable software components on a FACE/ARINC653 compute module whose scheduling was driven by a common clock. However, this subsystem communicated over a switched Ethernet with sensor and display equipment. These resources were not clock-synchronized with the compute module. Again, modeling guidelines and appropriate tool support need to be

provided for the AADL language features used to specify these aspects of systems. Finally, there is a need to support mixed fidelity modeling and analysis, e.g. our earlier discussion on "black box" modeling and analysis.

JCA DEMO ACVIP ANALYSIS FINDINGS AND LESSONS LEARNED

Previous studies have shown that peer review is a very cost-effective means of defect detection, partly because it was the only traditional method that could be applied in early development phases. The ACVIP researcher's experience is that many defects were detected during model development even before analysis tools were applied. This is achieved by mapping terms in the document into concepts expressed by AADL. Users quickly realize different terms used in different sections of the documents for the same concepts, and conflicting statements about specific attributes of model elements, e.g., two different numbers for range of operation. Strong typing in AADL ensures that interactions between virtually integrated system components are consistent, e.g., that measurement units and interchange protocols are used consistently. In other words, the rigor of the AADL focuses attention on ambiguous and incomplete elements of a natural language document and eliminates potential system integration problems early in the process. This is consistent with earlier reports that a significant benefit of modeling is more precise specification; many defects are found during the model development phase (Ref. 26).

Earlier studies showed that providing reviewers with structured guidelines (often called reading guidelines or techniques in the inspection literature) improved the quality of reviews. In model-based engineering, the model development task could be viewed as a particularly well-structured review method (Ref. 27)

The ACVIP related goals for JMR Mission Systems Architecture Demonstrations (MSAD) such as the JCA Demo are to identify, validate, mature and transition methods and tools to support an architecture centric virtual integration process. This exercise also generated new modeling guidelines and tool requirements (as well as bug reports for tool developers and errata for the AADL standards committee).

The ACVIP researchers provided reports citing around 85 findings, 70 that were attributed to requirements analyses and 15 to timing analyses that will be rolled up in the JCA Demonstration Final Report. Some notable areas identified by the ACVIP team included:

- Relationship of component states and MIS system state not being fully specified
- Lack of a specification of currency/staleness for the data
- No identification of end-to-end timing requirement for hazard data
- Partition schedule not meeting ARINC 653 scheduling rules
- Non-clarity in protocol from MIS to support multiple or single instantiation of DCFM
- Non-clarity in data storage requirement between the DCFM and MIS
- Ambiguity on the MIS system Operational State when a clock timer expires
- Lack of a requirement for the number of source tracks the aircraft survivability sensor provides
- Possibility of track jitter will be seen in integration
- Multiple sensor stream rates may have implications on integration.
- Cross partition timing issues in the ARINC 653 schedule
- Inconsistency in the area of threat ranges between the DCFM and MIS making it unclear how alerts would be handled
- Potential memory leaks in MIS identified
- Ambiguity in the requirement to correlate 50 source tracks within 1 second and concern over meeting the requirement.

Some of these issues with relation to the DCFM were also cited by the DCFM vendors independently of the ACVIP researchers. At the time of this paper's writing the MIS team were able to confirm several of these and other findings by ACVIP; however, several are still to be confirmed in integration testing. A spreadsheet of the findings by the ACVIP team was sent to the MIS team to confirm the findings. Some findings were dismissed by MIS because the identified issues had been addressed through the requirements adjustment made by MIS of which the ACVIP was not aware. In general, the findings by the ACVIP team demonstrated that in a real program that these issues would have been identified and corrected even prior to solicitation which could have led to a cost savings and / or development schedule reduction.

Outside of the issues directly affecting the DCFM and MIS integration, there were improvements identified in the OSATE tools, ACVIP Modeling and Analysis Handbook, FACE Generic Modeling Environment (GME) to AADL translator, improvements needed to mature the requirements, safety and timing analysis capabilities.

FUTURE MATURATION AND PLANS FOR ACVIP

The ACVIP metrics analysis and evaluations are still in progress at the time of writing of this paper. At the conclusion of the JCA Demo integration effort, retrospective analysis of defects detected in both the JCA Demo and the ACVIP Shadow effort will be completed. This will include estimating such things as when defects were detected and by what methods in both baseline demonstration and ACVIP shadow, and the costs and benefits of earlier error detection using the various ACVIP modeling guidelines and methods

and tools at various phases. These final evaluations will provide input into the plans and actions discussed in the paragraphs below.

The JMR program has developed roadmaps both for the development of advanced analysis approaches that leverage the integrated architectural analysis strength of AADL and the incremental analysis approach of ACVIP. New analysis methods will be added to the process incrementally as they emerge in the research community. These tools will be demonstrated to gain insight. Then tools and documentation will be matured to a point where third party developers in research oriented teams can effectively apply the tools. Handbooks for the technical use of the analysis methods and the ACVIP process as well as acquisition guidance for program managers are being developed and refined in each phase of demonstration. Tools to enable analyses for requirements, safety, security, resource utilization, timing, code generation and rapid integration are examples of JMR S&T focus areas to increase technology readiness levels (TRL). The capability to translate to/from other modeling languages such as UML and SysML to AADL is planned to be added with an attempt to translate the JCA Reference Architecture from UML to AADL as a first step. SAVI gains in tools, analyses, and processes will also be incrementally integrated into ACVIP. Furthermore, technology transition of the ACVIP processes and tools will occur through offered training and future JMR Mission System Architecture Demonstrations. It should be noted that an AADL/ACVIP training session as part of the JCA Demo ACVIP Shadow effort was conducted. The session included both industry and Government attendees. More training opportunities like this will be available in the future. These activities provide the Government and industry with guidance and experience using the AADL and ACVIP in preparation for FVL. The expected benefit is early discovery of integration issues throughout the development process reducing development cost, schedule and risks for FVL.

CONCLUSION

ACVIP is an architectural centric model based approach that will revolutionize the way in which we analyze our systems. Results of the JCA Demo ACVIP Shadow effort demonstrated that ACVIP has potential to provide strong architectural analysis to identify and aid in the eradication of issues. ACVIP and its guidance, tools, and processes are in its infancy and require further refinements and maturation to be effective for future DoD acquisition of aviation mission computing systems. AADL is being used in many company and organization research efforts and needs to be matured and transitioned to development and production areas. JMR Mission Systems Architecture Demonstrations will continue to work with the ACVIP researchers and ensure that the exercise, documentation and lessons learned mature these processes and tools so that they can effectively be used by avionics and systems engineers in the future. Industry and Government need to work together to improve ACVIP so that future development / integration efforts can benefit from early virtual integration, validation and verification.

REFERENCES

[1] Department of the Army, Army Contracting Command. "*A Joint Multi-Role Technology Demonstrator (JMR TD) Joint Common Architecture Demonstration (JCA Demo) Broad Agency Announcement (BAA)".* Location ACC-RSA-AATD-(SPS), 2014. Solicitation Number W911W614R000002.

[2] Aerospace Vehicle Systems Institute. http://savie.avsi.aero. [Online]

[3] NAVAIR. "*Technical Standard for Future Airborne Capability Environment (FACE)*", The Open Group, Public Release 2013-149.

[4] Standard, SAE Aerospace. "*AS5506 Architecture Analysis & Design Language (AADL)".* (ref. http://standards.sae.org/as5506b), Revision B 2012.

[5] OSATE2 (Open Source AADL2 Tool Environment). *AADL Public Wiki.* [Online] https://wiki.sei.cmu.edu/aadl/index.php/Osate_2.

[6] Hansson, Feiler and Helton, SEI and Boeing. "*ROI Analysis of the System Architecture Virtual Integration Initiative".* 2011.

[7] *A Survey of Architectural Description Languages.* Clements, Paul C. March 1996.

[8] Object Management Group. "*Systems Modeling Language (SysML)",* Version 1.3. [Online] June 2012. http://www.omg.org/spec/SysML/1.3/.

[9] SAE International, AS-2C. "A*S5506/2 Architecture Analysis and Design Language (AADL) Annex Volume 2: Data Modeling Annex, Behavior Annex, ARINC653 Annex".* 2011.

[10] ARINC. "*Avionics Application Software Standard Interface: ARINC Specifcation 653 Part 0".* June 2013. ARINC 653.

[11] Ellidiss. AADL Inspector. [Online] Ellidiss Software, 2012. http://www.ellidiss.com/products/aadl-inspector/.

[12] Dissaux, Pierre. *"Stood: a 'state of the art' hybrid real time software toolset"* . [Online] http://www.ellidiss.com/products/stood/.

[13] *"MASIW Framework: an open source Eclipse-based IDE for development and analysis of AADL models".* [Online] http://forge.ispras.ru/projects/masiw-oss.

[14] CMU-SEI. *"Summary of AADL Related Toolsets",* [Online] https://wiki.sei.cmu.edu/aadl/index.php/AADL_tools.

[15] Defense, Department of. "*Operation of the Defense Acquisition System".* 7 Jan 2015.

[16] *IEEE 830-1998, "Recommended Practice for Software Requirements Specification".* June 2009.

[17] DOT/FAA/AR-08/32. "*Requirements Engineering Managmeent Handbook".* June 2009.

[18] Association Française d'Ingénierie Système. CPRET: System Process as Constraints, Products, Resources, input Elements and Transformations. [Online] http://en.wikipedia.org/wiki/Process_%28engineering%29#CPRET.

[19] CMU-SEI. Quality Attribute Workshop. [Online] http://www.sei.cmu.edu/architecture/tools/establish/qaw.cfm

[20] CMU SEI. Architecture Tradeoff Analysis Method [Online] http://www.sei.cmu.edu/architecture/tools/establish/atam.cfm.

[21] SAE International, AS-2C. *Architecture Analysis and Design Language (AADL) Annex Volume 3 Annex E: Error Model Annex, Draft.* Dec 2013. AS 5502/3.

[22] SAE International, SAE ARP-4761. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.* 1996

[23] M. González Harbour, J.J. Gutiérrez García, J.C. Palencia Gutiérrez, and J.M. Drake Moyano. *MAST: Modeling and Analysis Suite for Real Time Applications.* http://mast.unican.es/, 13th Euromicro Conference on Real-Time Systems, 2001.

[24] Boddy, Mark. *Separation Platform for Integrating Complex Avionics (SPICA). SBIR phase I Technical Report.*: Adventium Labs, September 2013.

[25] Rob Edman, Hazel Shackleton, John Shackleton, Tyler Smith, Steve Vestal *"A Framework for Compositional Timing Analysis of Embedded Computer Systems"*, Adventium Labs, Feb 2015

[26] Edmund M. Clark, Jeannette M. Wing, "*Formal Methods: State of the Art and Future Directions", ACM Computing Surveys.* 1996.

[27] Laitenberger, Oliver, "*A Survey of Software Inspection Technologies, Handbook on Software Engineering and Knowledge Engineering".* 2002.