System Engineering Approaches for Performance Critical Avionics Embedded Computer Systems
Using the Architecture Analysis and Design Language (AADL)

Bruce Lewis
Computer Engineer, Experimental Developer
US Army
Redstone Arsenal, Alabama
bruce.a.lewis@us.army.mil

Abstract: System engineering practice for the design and development of the embedded computer systems in avionics today is significantly more empirical (experimental) than analytical. The result has been typically expensive problems in the integration and evolution of complex systems, whether the program succeeds or fails. The AADL provides a standards-based way to describe the computer architecture hardware and software components in precise terms allowing early analysis and incremental system construction to predictive models. Multiple domains of engineering analysis can be integrated into a single specification. It can be used to bridge system development phases. This new standard-based capability is foundational for advancing our ability to engineer avionics systems early and throughout the lifecycle across the industry. This paper overviews the history, standard, language, analysis methods, advanced architecture research programs using the AADL, provides large model examples, and includes publicly briefed slides from a NATO study on the rapid integration of weapons onto aircraft platforms that recommended the AADL.

Introduction
As computer based systems have become more complex and as we continue to exploit the benefits of code generation for components, the problem has become the integration of components and the emergent behaviour of the system. It’s not enough to have correct code for the software components or subsystems, they must be properly integrated and correctly executed to have a fully functional system that meets its performance (performance defined as non-functional qualities like latency, timing, scheduling, utilization, reliability, security, safety etc.) critical requirements. The problems have been compounded in complex avionics systems by complex integrations of multiple layers of scheduling, multiple forms of communication, multiple approaches to system execution control, plus the typically complex fault tolerance, security and safety requirements The problem domain is also multi-dimensional because the system performance qualities that must be achieved are highly cross coupled. For example, a change to a software component or the substitution of another hardware component can affect system latency, safety, fault tolerance, bus utilization, processor utilization, etc. And failure to meet the performance critical qualities required results in failure of the system just as surely as a functional component’s failure to provide the right output.

Yet, we as an industry had not developed a sufficient system engineering approach for these computer systems, preferring instead to test and simulate, or to over specify resources. But the end result has been that system integration for complex embedded or performance critical computer systems is one of the highest areas of program risk. It’s also a major cost driver on programs that do succeed. Program failure is too often the result of integration failure and may mean not only the loss of billions of dollars but also the loss of a system in our inventory for which we have invested many years as well as dollars.

Integration and re-certification issues have a major impact on the cost of upgrading. Upgrades typically require significant schedule for small changes with high re-certification costs. The ability to architecturally separate components and validate that separation simplifies re-certification. ARINC 653 is an example if its acceptance. What can be done to build more powerful architectural containers? Lower cost and more powerful architectural approaches are under investigation in research studies using the AADL. This objective will require advanced architectural analysis and validation capabilities.

As an industry, to meet these needs to precisely specify and model embedded performance critical systems, we must have a common standard language with strong semantics that can capture both the structure and dynamics of embedded systems. It must support the capture of properties of these systems and the analysis approaches to evaluate the critical qualities. It must be incremental to
support all lifecycle phases from early abstraction to final implementation and evolution. Since our analysis approaches will differ and grow, we need a language that is also extensible in a controlled fashion to preserve the benefits of a standard, but also provide flexibility.

The AADL was developed for just such a purpose. It was developed from significant experimentation and research over 15 years. It provides a language that is useful across domains where real-time, embedded, fault tolerant, secure, safety critical, software intensive systems are developed. Its natural fields of application include avionics, automotive, autonomous systems, industrial, medical, etc, but it should be considered in other performance critical domains.

The Society of Automotive Engineers (SAE) Architecture Analysis & Design Language, AS5506 [1], provides a means for the formal (analyzable) specification of the hardware and software architecture of embedded computer systems and system of systems. It was designed to support a full Model Based Development lifecycle including system specification, analysis, system tuning, integration, and upgrade over the lifecycle. It was designed to support the integration of multiple forms of analyses and to be extensible in a standard way for additional analysis approaches. It supports automated system integration via tools from AADL models when fully specified and when source code is provided for the software components.

History
The AADL language has been developed with input from three major areas. Its birth was multiple DARPA research programs. The proof of concept and base for development was the MetaH language. MetaH was developed by Honeywell Labs, with Dr. Steve Vestal as principle investigator, over 12 years and three DARPA programs [2]. It was used in over 40 experimental projects, many of them DARPA programs, internal Honeywell investigations, Army experiments and/or SEI experiments. From these experiments in multiple domains of application, but primarily avionics and flight control, over 30 improvements were defined for the next generation language.

Several of these projects were performed in our lab, one being the development of a reference architecture for missile systems and re-engineering a missile system to that architecture, building it and the 6DOF real-time simulation it flew in, software-in-the-loop, with MetaH and the MetaH toolset, including specification, analysis and automated system integration [3, 4]. Significant cost savings were obtained in development, and system and component changes were rapid and efficient including hardware upgrades, system partitioning, component partitioning and upgrades, rate changes etc. See figure 1 below.

Several experiments involved rapid development of components via Matlab / Simulink with Beacon code generation and rapid integration, to a MetaH specified system architecture, onto the embedded hardware using the MetaH toolset. These experiments and the proprietary nature of MetaH convinced the author that an open Architecture Description Language (ADL) supporting model based development of embedded systems was needed and was worth pursuing.

The second major area of input to the language was the other ADL languages developed by DARPA and within industry. See figure 2. Experience with these languages and MetaH, especially at the SEI by Dr. Peter Feiler, were also leveraged in the design of the AADL which broadened the domain of application and helped form the core AADL from which extensions would be later developed. Expertise on the standardization committee with UML and HOOD/STOOD were also leveraged to validate concepts, provide industrial strength solutions and ease integration within the industry. Coordination began with the Object Management Group over 3 years ago to develop a standard AADL Unified Modeling Language (UML) profile to provide the benefits of AADL to the UML community. The standardization of this profile is being done in partnership with the UML community.

Figure 1 Early DARPA Sponsored Experiment
Committee

The third major area of input was the SAE committee (see figure 3) which developed the requirements document, the core standard AS5506, and the annexes to the standard. Many language features were formed from the expressed needs of industry representatives from many of the major aviation and real time systems companies in the US and Europe. Many of the participants are leading engineers in their companies developing the next generation approaches for mission critical computer system development. These engineers recognized early the need for a common standard ADL to support computer system engineering in performance critical systems.

Co-ordinations with new research programs are already in place and developing, which will drive future capabilities to be added to the standard. See section on Significant Research Programs. There are also co-ordinations with NATO, other SAE committees, the Object Management Group, US Army, AF, and Navy, and European Space Agency.

State of Standard

The core AADL standard [1], version 1.0 was published in Nov. of 2004. A standard set of Annexes was published in June of 2006 [5]. Several new annexes are now under prototyping. One of these, the AADL UML profile has been informally balloted and will start formal ballot in March 2007.

AS5506 Core AADL language standard (Nov 2004)

– Textual language, semantics

AS5506/1 Annexes (June 2006)

– Graphical AADL Notation Annex
  Enables graphical AADL programming
– AADL Meta-model/XML Annex
  Model interchange & tool interoperability
– Programming Language Annex
  Mapping to Ada, C/C++
– Error Modeling Annex
  Dependability and fault modeling

UML Profile for AADL (2007)

  Enabling use in UML community
Behavior Annex (2007)

  Detailed component behavior modeling

Annexes provide a way of extending the language incrementally so that tools can support uniformly those aspects important for their domain of application. The core textual language, meta-model and XML schema and AADL graphics are being supported now in AADL tools. Standard error modeling capabilities that have been developed are now under experimentation along with
integration to dependability analysis tools. Specific tools will be discussed later in the paper. Version 2 of the standard is in process and will include new core capabilities. It is now largely defined.

**AADL Overview**

The AADL provides components with precise semantics to describe computer system architecture. Components have a type and one or more implementations. Software components include data, subprogram, thread, thread group and process. The hardware components include processor, memory, bus and device. The system component is used to describe hierarchical grouping of components, encapsulating software components, hardware components and lower level system components within their implementations.

Interfaces to components and component interactions are completely defined. The AADL supports data and event flow, synchronous call/return and shared access. In addition it supports end-to-end flow specifications that can be used to trace data or control flow through components.

The AADL supports real-time task scheduling using different and extendable scheduling protocols. Properties to support General Rate Monotonic Analysis and Earliest Deadline First are provided in the core standard. The core also provides a property extension construct to define properties needed for additional forms of analysis. Execution semantics are defined for each category of component and specified in the standard with a hybrid automata notation.

Modal and configurable systems are supported by the AADL. Modes specify runtime transitions between statically known states and configurations of components, their connections and properties. Modes can be used for fault tolerant system reconfigurations affecting both hardware and software as well as for software operational modes.

The AADL supports component evolution through inheritance, allowing more specific components to be refined from more abstract components. Large scale development is supported with packages which provide a name space and a library mechanism for components, as well as public and private sections. Packages support independent development and integration across contractors.

AADL language extensibility is supported both through a property construct for specifying or modifying AADL properties and an annex extension mechanism that can be used to specify sub-languages that will be processed within an AADL specification.

Annexes will often also provide additional properties for a domain of use. An example is the Error Modeling Annex which allows specification of error models to be associated with core components. See section on error modeling.

A number of other papers are referenced for a more detailed description of the language [6-8].

**AADL Examples**

To provide a better feel for the language I’ve included AADL specifications from publicly available AADL model of a helicopter system. The following AADL text describes the flight manager. This subsystem is a partition in a larger AADL specified avionics system and is described in AADL using an AADL process component consisting of a flight manager type and a flight manager implementation. An instance of the flight manager will be created when it in turn is used as a subcomponent in a higher level component’s implementation.

The example code for the flight manager process implementation (what is contained in the flight manager) and one of its thread’s type specification, NavigationSensorProcessing, is provided below. The type specification provides the component’s interface (features) and some properties that would be used in all implementations of this type. In the example below, each of the thread’s properties are pre-defined in the language. The Dispatch_Protocol thread says that this is a periodic thread; Period says its period is 50 milliseconds. The Compute_Execution_Time property says that when dispatched it will execute for 5 to 15 milliseconds. If fault handling is implemented, it would raise an error if it exceeds 15 milliseconds. There are many pre-defined properties in the standard and also a mechanism for users to define their own properties.

The flight manager is an AADL process, that is a protected address space, and in this case, will have a property indicating that it is a partition in the application (protected for maximum execution time) and a property for its partition execution time. Today, these would be user defined properties but in version two they would become standard. Such properties would typically be inserted at the process’s type level since they would logically hold for all implementations. If re-defined in the implementation, they override the property defined at the type level.

The process implementation lists all the subcomponent instances that it contains (it could also be defined as an extension of other implementations and just include new subcomponents), their connections (some were omitted in the code below), and one of its defined flows for latency analysis. Notice that the instance name for the NavigationSensorProcessing thread is NSP. For the instance, properties can be re-defined.
The threads are connected by connections between ports. The process has port groups which group input and output ports that attach to contained subcomponent ports, in this case, threads. The different port categories, data, event, event data, have specific semantics. The connections between ports also have specific semantics, in this case data ports with immediate and frame delayed communication to give deterministic behavior.

The flow path specification in the example is defined in the implementation to show the path from input port to connection to component to connection to ... to ultimately the output port. It will be used with other flow paths and a flow source and flow sink to specify an end to end flow within the system.

Modes and mode transitions have not been included in this example implementation but can be used to indicate alternative implementation configurations for execution which will be triggered by an event to the component.

**thread** NavigationSensorProcessing

**features**

- navSignalDataFromSensor: *in data port*
  
- navSensorDataToIN: *out data port*

**properties**

- Dispatch_Protocol => Periodic;
- Period => 50 Ms;
- Compute_Execution_Time => 5 Ms .. 15 Ms;

**end NavigationSensorProcessing:**

**process implementation** FlightManager.PIO

**subcomponents**

- NSP: *thread* NavigationSensorProcessing;
- INav: *thread* IntegratedNavigation;
- GP: *thread* GuidanceProcessing;
- FPP: *thread* FlightPlanProcessing;
- APC: *thread* AircraftPerformanceCalculation;
- pageFeed: *thread* HandlePageRequest;
- PerIO: *thread* PeriodicIO;

**connections**

- navSensorconn: *data port*
  
- navdataconn1: *data port*
  
- navdataconn2: *data port*

**flows**

- cmd_request: *flow path* FMToPCM ->
  
- inflow: *port group* FMIn -> PerIO.fromOutside;
- outflow: *port group* PerIO.toOutside -> FMOut;
- fuelflowconn: *data port* PerIO.fuelFlow ->
  
Figure 4 shows the graphical specification for the flight manager implementation. This graphic was not produced from this code example so it lacks the HandlePageRequest thread for which the flow is defined in the above example. Here you can see the graphical symbols for the threads, the data ports, the port group, and the immediate and delayed connections.
Decorators on the threads show the thread period and can also show the thread property relative to its dispatch (periodic, aperiodic, sporadic, and background). The execution behavior of each of the AADL component categories is precisely defined using a hybrid automata notation in AS5506. This definition is formal so the semantics can be precise. The definition includes normal and recovery, fault handling, resource locking, mode switching, initialization and finalization behavior. See Figure 5.

Figure 5 AADL Hybrid Automata Definition

Figure 6 below shows a complete flow path specified at the component type level from a flow source in a device through connection to a system defining a flow path across its input to output ports through a connection to a flow sink defined in a device. The implementation of the system component would then define the flow path based on the subcomponent composition and connections of that component, which was part of the code example. At the type level, the flow related properties could specify the required flow timing. At the implementation level, the actual flows, estimated or measured, could be captured and used to validate the flow requirement specified at the type level.

Figure 6 End to End Flow Specification

Analysis Methods

Once an architecture or a critical aspect of the architecture has been captured in an AADL specification, the properties can be added for the analysis of interest. For analysis, architecture data is extracted from the specification by plug-ins (in Eclipse based OSATE environment, see section on tools) and are solved within the plug-in or exported to an external analysis toolset. These analyses can be incrementally added to a specification by adding new properties and adding any additionally needed aspects of the architecture. Figure 7 below shows domains of analysis prototyped and demonstrated with the AADL. Each of the listed methods within a domain may have multiple analyses forms. Analysis development has been a rapidly expanding area. Research continues to expand the available tools.

AADL supports multiple incremental forms of analysis which allows integration of results

Figure 7 Capturing Multiple Critical Qualities

Here are some example analyses. The latency analysis example provided by Rockwell [9] is based on a helicopter architecture and shows the OSATE toolset and its output.
indicate a violation in the real-time performance domain. 1600 flows were specified. See figure 8 below.

Results in the big picture

Model and validate an architecture with respect to data quality attributes

- Determine an architecture is secure, i.e., it does not compromise confidentiality and integrity, and that sanitization can be performed correctly
- Validate that applications under different modes (scenarios) use data of sufficient quality, e.g., compare normal operating mode, failure mode, and overload mode scenarios
- Modeling experience from validating real sensor network applications
- Analysis tool for performing target detection using a sensor network and for determining communication bandwidth schedulability

Figure 9 Data Quality Analysis, Architecture Validation

Project Status: Proof-of-concept Prototype

<table>
<thead>
<tr>
<th>Modeling Areas</th>
<th>Skills Required</th>
<th>Skills Analysis</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>Delivered</td>
<td>Confines and discrete</td>
<td>Documentation in the work</td>
</tr>
<tr>
<td>Confidential</td>
<td>Delivered</td>
<td>Confidential and discrete</td>
<td>Confidential and discrete</td>
</tr>
<tr>
<td>Security</td>
<td>Delivered</td>
<td>Confidential and discrete</td>
<td>Confidential and discrete</td>
</tr>
<tr>
<td>Consistency</td>
<td>Delivered</td>
<td>Documented in TN &amp; Criteria</td>
<td>Documented in TN &amp; Criteria</td>
</tr>
<tr>
<td>Sensor networks</td>
<td>Delivered</td>
<td>ANDES platform</td>
<td>ANDES platform</td>
</tr>
</tbody>
</table>

Figure 10 Deliverables include AADL Methods and Tools

Analysis in the resource consumption domain on a helicopter system was demonstrated at Rockwell using internal analysis tools. The analysis results identified the possibility of reducing the number of processors, thereby decreasing system cost on each system. See Figure 11. Analysis of resource consumption using AADL is available through multiple toolsets. See tools section.
Model Based Development with AADL

There are multiple processes that might be used in a Model-Based Development. The key concept is the analysis of the computer based system specification and then implementation in accord with the models. The operational architecture, the hardware and software system that will be executed and its performance qualities, is the target of analysis. The process can start from many points as shown in figure 12. For instance it might be used to capture a product line architecture and its variants so that performance of any instance can be evaluated and the whole family managed to maximize reuse within the constraints of the basic architecture. Model-Based Development should fit within and add to typical development approaches.

A model based approach can start from a legacy system being evolved, a system in development, or a new development. It can be driven by identified architectural risks. It can be integrated with various other engineering capabilities, especially those related to code generation of components, such as Simulink, or higher level more abstract design approaches, such as UML. AADL annexes like the UML, Behavior and ARINC 653 which are in process of development within the standardization committee, will define common mappings for use with various methods and within application domains. The error modeling annex exists now.

Based on architectural artifacts, the system is captured to the fidelity needed for the performance analysis required. Architectural pattern analysis can be used to evaluate how the system should be represented and what types of analysis might be needed. Later phases can then develop detailed models and add more analysis approaches.

Since the AADL may relate to multiple activities and system databases, the AADL provides a foundational semantic base for exchange between tools in a development process. In effect, system repositories should allow simple extraction of architecture models that are up-to-date in the development process or as-built. Detailed models then can be rapidly constructed through extraction and generation. This approach is being demonstrated at Rockwell Collins [9]. See figure 13.
of the architecture as the architecture itself is refined and as software components are developed and refined. Multiple architecture analysis methods, such as scheduling, latency, safety, are selected and run on the system model as it is incrementally developed. Software component source code is supplied by the user and can be generated from component generators, hand coded, or reused/re-engineered. Given the AADL specification, source code for the application and the execution environment (processors, buses, memory, devices plus operating system, compilers etc), tools can automate the process of system configuration, composition and runtime system generation. These system integration/generation tools need to be consistent with the analysis methods and AADL semantics—generated according to the AADL specification. See Figure 14.

Basic Methodology

The Model Based Development process we have used in our laboratory in the past with MetaH includes the concepts of architectural specification, architectural analysis, and automated integration with generation of communication, glue code and the system executive to construct the final system. The AADL supports all the MBD concepts of MetaH and adds significant additional capability and flexibility in a public standard.

Architecture analysis is run each time the specification is updated, to provide model checking
be accepted by OMG as a standard profile. This profile is currently based on UML 2 but it is anticipated that as MARTE emerges from the OMG, that a MARTE based profile will also emerge. We are working with the MARTE committee to assist in its definition. This approach allows designing in AADL components within a UML toolset environment.

Another approach for use of the AADL with UML is being developed. This approach was recommended and documented in a NATO study [13]. See Figure 15.

The integrated use is illustrated in Figure 17. The xUML provides the reusable platform independent model and it is translated into AADL. The AADL provides the PIM specification, analysis, system tuning, binding, and can also provide automate integration. Issues discovered at the AADL level can be fed back for adjustment either in the xUML model or the AADL model.

This integration seems to be a natural, given that the AADL is oriented to the Platform Specific Model (PSM) and xUML to the Platform Independent Model (PIM). Note that the abstractions in each language clearly support the model domains of use. See Figure 16, provided by Chris Raistrick, who served as a consulting member of the NATO study group.

The AADL would be used to generate the glue code to its specifications for integration. One of the sources for software component generation could be the xUML action language. A translation approach has been developed and Chris has developed a prototype generator of AADL specifications from xUML specifications.

Of course a significant amount of platform dependent information must also come in to the AADL specification to provide the AADL model with the PSM detail needed. The amount of information will depend on the analysis approaches to be used and can be incrementally expanded as new analyses are added. This information can come in through design and “as-build” databases on the systems being integrated, in the case of ALWI-CL, the platform, launcher and weapon. Such a toolset prototype has been developed to provide, from AADL defined data, table driven generation of AADL specifications. See figures 18.

Figure 15  xUML/AADL Approach from ALWI-CI

Figure 16  xUML and AADL Modeling

Figure 17  xUML and AADL Process
The standard components of the AADL and its domain specific nature, which expresses the way systems are built in the domain, make this approach natural and generally useful for the rapid capture of the architecture of complex systems.

The Ellidiss toolset, STOOD [14], provides some transformation capabilities for integrated use of UML and AADL. See section on tools.

**Integrated development with Simulink**

Strong interest has also been expressed for integration with Simulink from a number of AADL users. In our experiments developing flight controllers for an aircraft and for a satellite system, we developed the architecture desired in the AADL, then the control system in Simulink using the component abstractions in the AADL architecture. These control components were then generated from the Simulink specification using Beacon with a special switch to allow generation for integration in the MetaH toolset (pre-AADL).

The AADL Programming Language annex [15] which provides guidance for building/integrating AADL compliant systems in C and Ada, should be useful for integrating components from various component generators. Several AADL tools have already demonstrated code integration/generation capability and are listed under the tools section. We are considering a new annex for guidance on system generation.

**Tool Strategy and Progress**

The SEI, via Dr. Peter Feiler, the principle author of the AADL standard, developed the Open Source AADL Tool Environment (OSATE) [16]. It was developed in the Eclipse framework using the meta-model standardized for the AADL. It provides the standardized XML definitions for use by Eclipse AADL plug-in analyses or interfacing with external toolsets. OSATE is available at [http://www.aadl.info](http://www.aadl.info). This toolset has been in use since the release of the standard in Nov, 2004.

The Eclipse tool framework and the AADL tools developed within it provide significant power for the development or integration of analysis tools. Analysis plug-ins are much easier to write and customize than stand-alone toolsets. They focus on the extraction of data and its analysis with methods provided within OSATE for traversing the XML expressed models. Existing toolsets can be integrated into Eclipse or interfaces to tools developed in Eclipse. Stand-alone tools can also be generated from Eclipse plug-ins. See Figure 16 above.

There were multiple reasons for an open source toolset strategy. One was to accelerate availability of commercial tools by providing an open source tool that fully incorporates the language and provides semantic as well as syntactic checking. OSATE also has the benefit of making the full language available from the beginning, and validating the language standard itself through implementing the language. It also furnishes a low entry cost vehicle for getting started with the language and provides a vehicle for in-house prototyping for the development of analysis approaches or annex extensions. It has accelerated use by early adopters of the language. It decreases the likelihood of incompatible toolsets implementing their own flavor of the standard.

Since the AADL provides several standard extension mechanisms, industry domains, companies, and vendors can provide extensions through these forms without corrupting the standard. These extensions then can be submitted to the AADL standardization committee for consideration as part of the standard. An example of this is the Airbus developed Behavior Annex [17] which is now being reviewed by the committee.

Commercial tools are highly valued by industry for their development process support and on-call maintenance. The UML profile is being developed to make it easy for UML tool vendors to support the AADL.

Commercial analysis tools may be integrated with commercial or open source AADL tools and AADL analysis tools themselves present an opportunity for commercialization. Current commercial design tools can also be modified and extended to support the AADL. For instance, Ellidiss (TNI Europe Limited) has extended their commercial STOOD HRT-Hood, UML 2 development environment to support AADL. It provides a number of Integrated Development Environment (IDE) features. See tools section.

The AADL meta-model annex [18] includes the AADL Declarative Model and the AADL Instance model (see Figure 18 XMI Based Tool Integration).
The XML/XMI expression of the declarative model can be converted back into a textual specification or graphical specification of a system. The Instance model is used to simplify analysis by providing the system instance as it will be bound to the hardware resources with all inherited properties. The meta-model provides a specification of the language that can be used in meta-modeling frameworks to rapidly build AADL compliant toolsets, such as has been done in TOPCASED [19, 20] or GME [21].

The XMI schema provides a database for analysis tool interaction that is standard so different tools all use the same interface. Tools can not only read from the XMI models but also post back into it. For instance, a scheduling and binding toolset would be used to optimize processor and bus utilization given the constraints on binding captured in the AADL and the properties of threads and communication overheads. Given that binding, a reliability, latency or safety analysis could follow. Finally, from the system instance, the system could be automatically integrated with generation of glue code. See Figure 18 above.

**AADL Tools Available**

Some AADL tools are internal to the companies developing them. AADL users are developing interfaces to their internal analysis tools. Others are open source or commercial and more are expected from various research projects. Here is a partial list of what is available now (Feb. 2007).

- **OSATE [16]**
  - IDE – Open source, integrates textual, object modeling (XML), and graphical editors with semantic checking and consistency management across specifications. Uses TOPCASED AADL graphics. Integrates error modeling and behavior annex sublanguages. Integrates plug-in analysis tools
  - Analyses –
    - End to end latency across partitioned and non-partitioned systems,
    - Resource budgeting and utilization for processors, buses, power.
  - Scheduling for EDF and RMA.
  - Binding optimization (bin packing) for processors and buses.
  - Security, confidentiality, need-to-know, constancy, sanitization, and integrity
  - Data confidence, correctness and validity
  - Various architecture checks for correctness and completeness
  - RapidRMA interface and properties

- **TOPCASED [19, 20]**
  - IDE – Controlled open source with quality strategy for use in aviation industry. Airbus led, 30 partners, funding 20 Million Euros, Eclipse based, meta-modeling and transformation framework, includes AADL OSATE toolset and AADL graphics, AADL XML, model transformation tools and multiple other modeling notations including UML. Adding documentation and requirements tracing. Version 1.0 July 2007.

- **OCARINA [22-24]**

- **Fremont Furness Toolset [25]**
  - IDE - Open source with maintenance support. Builds on OSATE and TOPCASED to provide better user interface. AADL validation test suite.
  - Analysis - AADL to ACRS [26] (process algebra), formal analysis of concurrent resource utilization, scheduling
  - Generation - AADL to Charon [27], generation of control components and integration of components for hybrid control systems to AADL specification using Charon annex.
Simulation – simulation of AADL specification integrated with process algebra capability.

- **GME [21]**
  - IDE - Vanderbilt Univ, DARPA sponsored meta-modeling framework, supports custom domain specific languages with generation of toolsets. Supports AADL graphical architecture specification.
  - AADL security analysis, intrusion
  - Generation - From AADL specification and security properties generates network distributed RT system with security kernels on each processor

- **CHEDDAR [28]**
  - Analysis – ENST Bretagne, Open source advanced scheduling analysis toolset with AADL properties predefined for supported analysis approaches via capture in AADL specifications.

- **RapidRMA**
  - Analysis - Commercial advanced scheduling analysis toolset. SEI has developed OSATE plug-in to export data into Rapid RMA and AADL property set based on Rapid RMA concepts.

- **STOOD [14]**
  - Generation - to Ada, C++ or C templates from AADL specifications, provides reverse capability from Ada C++ or C to AADL.
  - Analysis – integration with Cheddar, TOPCASED and OSATE.

- **Axlog [29]**
  - Simulator – Open source, simulated execution of AADL specification. Will support behavior annex and can support user annexes. Will provide maintenance & customize.

Additional AADL tools currently in development that we know about: xUML to AADL translator (Kennedy Carter), Dependability analysis toolset integrations to Mobius (UIUC) and Sharp (Emery Riddle), AADL UML profile to AADL XMI translator (SEI), Table driven AADL generator (SEI), Aircraft level simulation of buses and weapon integration (21st Century), Sensor network performance and dependability (ANDES – UVA) [11]. These do not include proprietary analysis tools being developed within aerospace companies.

**Error Modelling Annex Concepts**

In fault tolerant, safety critical systems error modeling is an important aspect of architectural design and should be integrated into the architecture specification so it can be cross checked against changes. The MetaH language originally supported reliability modeling and this has been significantly extended in the AADL to support multiple forms of safety and dependability analysis through error models that are attached to architectural components [12, 30]. See Figure 19.

![Composing Component Error Models](image)

Figure 19 Component Error Models

Each component can have an error model. The architecture specification provides the foundation for understanding propagation and the effect of failure. Component error models integrate into a system error model through the architecture specification. Architecture specification changes then affect the error modeling..
directly making it much easier to maintain and discover impacts. See Figure 20.

Some of the supported analysis approaches include hazard analysis, failure modes and effects analysis (FMEA), fault trees, and Markov processes. Honeywell has demonstrated error modeling (hazard and FMEA) using annex capabilities on a large aircraft system [31]. ASSERT has modeled a dual redundant fault tolerant computer system using the annex [32, 33].

Error models are used to develop multiple types of system dependability models and these can be analyzed within a single specification allowing consistency checking between them. See figure 21.

Some of the early adopter presentations on the AADL are highlighted in the slide in Figure 22. These presentations and many others are available on the AADL website www.aadl.info. They include experimentation using the AADL to capture a reference architecture for military aircraft (EADS [34]), the development of a system engineering process using the AADL for validation of correctness and integration of the dynamic and structural aspects of the aircraft computer system (Airbus [35]), a presentation on an integrated UML and AADL process for the development of weapon system Plug & Play (PnP) architectures (General Dynamics [36]). Also included is a presentation on the modeling of a large modern aircraft system with architectural trade-off analysis, scheduling and safety analysis (Honeywell [31]), Figure 23, and a modelling and analysis of a modern helicopter architecture for system workload, partitioning and tuning of the switched network (Rockwell [37]). See Figure 24. Also pictured is a presentation on the development of a system engineering approach using the AADL and a formal methods oriented development process (ESA, ASSERT [38]). These presentations demonstrate a variety of uses, integration into system engineering processes and application to significant modern, complex, large, performance critical systems for the AADL.

Figure 20  Architecture Integrates Error Models

Figure 21 Integrating Dependability Analysis

Figure 22 Demonstrations of AADL Capability

Figure 23 provides a list of aircraft that Honeywell has modelled using MetaH/AADL [12]. This presentation also provides Honeywell’s tool development strategy for AADL tools.
Evaluations

Evaluations of various methods and tools have been carried out over the past few years using one or more of the following workloads.

**Air transport aircraft IMA (simplified production workload)**
- Globally time-triggered
- 6 processors, 1 multi-drop bus
- 105 threads, 51 message sources

**Military helicopter MMS (first release, partial)**
- Globally time-triggered
- 14 dual processors, 14 bus bridges, 2 multi-drop buses
- 306 threads, 379 [source, destination] connections

**Air transport aircraft IMA (preliminary, partial)**
- Globally asynchronous processors, precedence-constrained switched network
- 26 processors, 12 switches
- 1402 threads, 2644 [source, destination] connections

**Regional aircraft IMA (production workload)**
- Globally time-triggered
- 49 processors, 2 multi-drop busses
- 244 processes (TBD threads), 3179 [source, destination] connections

Figure 23: Honeywell MetaH/AADL Aircraft Modeling

**Rockwell Collins Large Proof of Concept**
- See full presentation on AADL web site
  - Generic Display System with Rockwell Collin’s Switched Ethernet LAN
    - Only LAN-related entities modeled
    - Model generated from Input/Output & Thread data stored in Database
  - Model Size
    - 6 Common Processing Modules (Processors)
    - 13 Virtual Machines (Partitions)
    - 96 Threads
    - 165 End-to-end Data Flows
  - 22,000 lines of AADL generated
  - OSA TE can handle 35 copies with reasonable performance on laptop, 700,000 lines

Figure 24 Rockwell Proof of Concept

Significant Research Programs Drive Future Capability

The AADL standard’s capability and extensibility has attracted a number of research projects on embedded systems analysis and verification. As was discussed in the section on analysis methods, some leading edge analysis work is also being done in Networked Sensors. See Figure 25 below. Each of these research projects is cooperating with the AADL standardization committee to identify new capabilities needed in the language as well as develop analysis and code generation tools, and engineering methods.

The ASSERT project and the TOPCASED project have already made significant contributions to the language and tools. The Networked Sensors project may be starting some momentum in the area of networked systems analysis, and certainly will assist in identifying AADL annex extensions for netcentric systems.

Several are focused also on new approaches to system verification and validation as well as certification.

European research projects are much larger both in dollars and participants. Both the European Space Agency and Airbus identified the AADL as a critical and foundational capability and have been working with the AADL since prior to its standardization in 2004. Some European projects involve co-investment split between the company and government. The ASSERT project is 15M Euro, TOPCASED 20M Euro, and SPICES 16 M Euro. Each of these has a significant focus on the AADL. The SPICES project has the strongest focus on the AADL to-date and is also the newest European project.

The Support for Predictable Integration of mission Critical Embedded Systems (SPICES) project [39], as described in their presentation to the AADL committee, is strongly oriented to AADL modeling and analysis for use in containers which would be integrated to form systems. “The principal goals of SPICES are to extend the capacities of the microCCM component-based framework and to couple it with an AADL modeller in order to offer to system architects, software architects, and application designers a component-based modelling, design and analysis environment for distributed real-time embedded systems that should be deployed over heterogeneous targets such as GPP, DSP or FPGA.” See Figure 26.
SPICES – Support for Predictable Integration of mission Critical Embedded Systems

Figure 26 AADL modeling and Container Concept

Figure 27 provides an overview of three of the work packages. The first is to develop needed extensions to the AADL. These typically would be annexes like the Behavior Annex Airbus is currently developing, but also corrections, extensions to the current standard where needed. The second work package is the development of AADL modeling tools and integrated verification and validation tools. The third work package is the generation of validated FPGA’s and validated software component containers and how to build tools to port them onto the platform. Two work packages not shown on the figure focus on case studies and dissemination.

Figure 27 SPICES Extends AADL through System Building

The newest research program with a strong AADL focus is sponsored under the Aerospace Vehicle Systems Institute (AVSI) System and Software Integration Verification (SSIV) committee. The AVSI is a cooperative of aerospace companies, DoD, and FAA to improve the integration of complex subsystems in aircraft. The AVSI SSIV co-located and met with the AADL standardization committee in Jan 2007. The information herein included was publicly presented at the meeting or cleared for publication.

Under a previous task the SSIV had selected the AADL as the ADL they were interested in using and extending. The theme of the new research is ”Integrate –then Build” and is a two phased program with a planned investment of 12 M dollars. The phase II project is expected to start in July of 2007 and end Dec. 2010.

This research project is still open to new members. It has 11 participants now and another 9 potential participants. Boeing and Lockheed Martin are spearheading the project. However participants are welcome from other performance critical domains, such as automotive, industrial control, space, etc. Each participant contributes three manyears of effort and pays a share of the administrative cost. Each participant shares in the results of all. Participation is considered essential to winning a position on the next major airplane program.

Three slides from the AVSI capture the problem, the approach and the solution. See figures 28, 29, and 30.

The Problem

- Airbus and Boeing have data that support Systems doubling in size and complexity every 2 years.
- Growing use of Integrated Modular Electronics and COTS
- Requirements continue to be refined throughout product lifecycle
- Integration costs increasing as systems do not function as “specified”
  - Unanticipated Interactions (Emergent behavior)
- Desire to design a new airplane every year or 2 instead of every 10.
- Desire to substitute subsystems on the airplane from different vendors. (Airline can make choice as is currently done on Engines)
- Desire to support incremental certification

Figure 28 The Driving Force for Change
The Approach

- AVSI did a previous study to determine a more exact way to define requirements supporting integration (2 years ago)
- The current proposal plans to further the previous study, extending a requirements /architecture /modeling language (i.e. AADL)
- Coordinate Component based, Model based and Proof based Development
- Defining standards on how to use it in the Aerospace domain
- Validating the approach with a pilot project.

Figure 29 Component-Based, Model-Based, and Proof-Based using AADL

The Solution

- The SSIV project will evaluate and define a solution that will allow developers to:
  – “Verify Requirements”
  – Predict and analyze functional, structural, behavioral, Timing and Utilization
  – “Integrate – then build”

Figure 30 Solution: Integrate – then Build

AADL Transition Support

The Software Engineering Institute is providing transition support for the AADL in the US and Europe as part of their Performance-Critical Systems group. They are the developers of the OSATE toolset and a training course and Guide document on the development of OSATE analysis plug-in development. They have developed a public two day AADL course on Model Based System Engineering with the SAE AADL and have published AADL research reports [40-43]. Three other AADL courses are in development. They have also developed a Practitioner’s Guide [44] on AADL and Control Systems Applications and a User’s Guide [45] to AADL notation. The SEI’s Performance-Critical Systems group will soon offer a service to do special analysis and architectural assessment using the AADL for performance critical system architectures.

Axlog in Paris, France is also offering AADL training. A number of universities have begun to involve PhD students in AADL research. Emery Riddle, Clemson, and ENST are offering courses using the AADL.

Conclusion

The AADL provides or supports through tools significant model based embedded system engineering benefits. These include:

- Precise semantics supporting analyzable models to predict system performance and drive development
- Prediction of system runtime characteristics at different fidelity
- Bridge between application engineer, architect and software engineer
- Prediction early and throughout lifecycle
- Reduced integration and maintenance effort

The AADL also provides additional benefits based on its standardized features including:

- Common modelling notation across organizations
- Single architecture model augmented with properties for multiple analysis
- Interchange & integration of architecture models
- Tool interoperability & integrated engineering environments

The AADL is based on over 12 years of research, over 40 experiments, other DARPA ADL’s and an expert committee. The AADL has been demonstrated on large complex embedded safety critical systems by leaders in the Avionics domain. A growing number of tools as well as transition support for AADL are becoming available.

System engineering for performance critical systems and system integration are two major issues for our industry. The research programs described demonstrate the need with their focus on AADL and concepts like “Integrate – then Build”, model-based development, component based architecture, pre-verification and validation etc. It is time to start using the AADL in your IR&D programs as evidenced by the growing research base employing AADL. As upgrades and new system developments occur, it is time also to investigate opportunities for application of AADL to enhance system engineering practice.

Acknowledgement

The author would like to thank the AADL standardization committee for their diligent efforts in language development.
development and their willingness to share through their presentations what they are doing with the AADL. For any interested in joining the AADL committee, they can contact the Chair, bruce.a.lewis@us.army.mil.

References

23. Hugues J. "AADL to build Distributed Real-time Embedded systems, experiments with Ocarina".


36. LaCerte Y. "AADL and MDA – Early Experience Applied to Aircraft-Weapon Integration ". In: AADL Standardization Meeting; 2005 January; Seal Beach; 2005.


Glossary

AADL: Architecture Analysis & Design Language
ADL: Architecture Design Language
ASSERT: Automated proof-based System and Software Engineering for Real-Time systems
ENST: Ecole Nationale Superieure Telecommunications
TOPCASED: Toolkit In Open source for Critical Applications & SystEims Development