The SAE Architecture Analysis & Design Language (AADL) A Standard for Engineering Performance Critical Systems

Bruce Lewis US Army Aviation and Missile Research, Development & Engineering Command, Huntsville Al 35898

Peter Feiler Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA 15213, USA, phf@sei.cmu.edu

Steve Vestal Honeywell Technology Center, Minneapolis, MN, USA, Steve.Vestal@honeywell.com

Abstract: The Society of Automotive Engineers (SAE) Architecture Analysis & Design Language, AS5506, provides a means for the formal 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. A system can be automatically integrated from AADL models when fully specified and when source code is provided for the software components. Analysis of large complex systems has been demonstrated in the avionics domain.

Keywords: Architecture, Computer Architecture, Model Based Development, Architecture Design Language, Computer System Engineering, Computer Modelling, AADL, Architecture Analysis & Design Language.

1. 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. 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 critical requirements. The problem is multi-dimensional because the system performance qualities that must be achieved are highly cross coupled. For example, the 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 or an algorithm's failure to control adequately. 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 and integration issues have a major impact on the cost of upgrading.

A model based approach, where the models we use to analyse the system are the models that will drive its execution and communication provides a much more predictable and powerful approach. In such an approach designers engineer rather than populate with rough estimates and testing. As Dr. Eric Conquet of the European Space Agency and Director of ASSERT stated in a presentation [1], "Software crisis: origin is in fact a lack in system engineering " and "Use of formal techniques at the software level without any formal approach at system level is a nonsense".

As an industry, to meet this need 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. 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.

2. Standard Development

2.1 History

The AADL language has been formed from three major areas. The proof of concept and base for its 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.

The second major area of input to the language was the other ADL languages developed by DARPA and within industry. See figure 1. 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 an industrial strength solution 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.

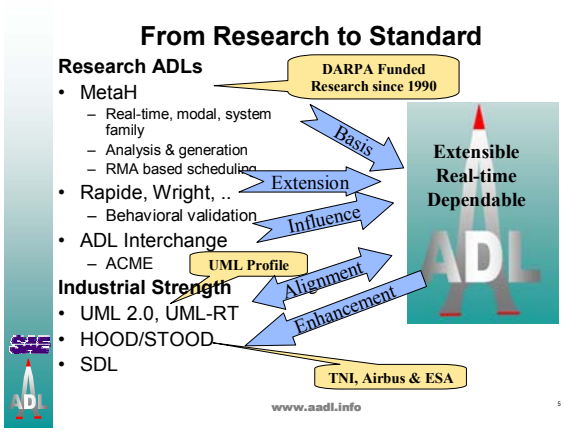


Figure 1 : Building from a foundation

The third major area of input was the SAE committee (see figure 2) 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 computer system development. These engineers recognized early the need for a common standard ADL to support computer system engineering in performance critical systems.

The SAE standard is being coordinated with North Atlantic Treaty Organization (NATO). NATO is also applying the AADL along with the US Air Force and another SAE committee, AS1, to develop approaches for rapid weapon system integration [5,6,7]. The AADL will be part of the OMG UML MARTE profile and the AADL committee has members on the MARTE committee. Research projects in Europe have significantly contributed to the standard, and are expected to continue to do so. COTRE [8,9] and TOPCASED [10] are two Airbus led programs that have worked closely with the committee. The ASSERT [1] program, European Union funded and European Space Agency led, is also working closely with the committee. These programs are applying the language, extending the language through annexes and developing tools.

Figure 2 : AADL Committee

2.3 Current State

The core AADL standard [11], version 1.0 was published in Nov. of 2004. Annexes have been or are in the process of being developed and balloted with the ballot dates shown in parentheses.

Annexes provide a way of extending the language incrementally so that tools can support or use those aspects important for their domain of application.

- Core AADL language standard (Nov 2004)
  - Textual language, semantics

- Graphical AADL Notation Annex (April 2005)
  - Enables graphical AADL programming
- AADL Meta-model/XML Annex (April 2005)
  - Model interchange & tool interoperability
- Programming Language Annex (April 2005)
  - Mapping to Ada, C/C++
- Error Modeling Annex (Oct 2005)
  - Reliability and fault modeling
- UML Profile for AADL (Mar 2006)
  - Transition path for UML practitioner community
- Behavior Annex (July 2006)
  - Detailed component behavior modeling

2.4 AADL Language

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 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 language 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 software operational modes.

The AADL supports component evolution through inheritance, allowing more specific components to be refined from more abstract component. 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 through a property sublanguage for specifying or modifying AADL properties. The AADL also supports extensibility through an annex extension mechanism that can be used to specify sub-languages that will be processed within an AADL specification. An example is the Error Modeling Annex which allows specification of error models to be associated with core components.

A number of other papers are referenced for a more detailed description of the language [13,14,38].

3. Model Based Development and Process Integration

3.1 Model Based Process

The Model Based Development (MBD) 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. See figure 3 for the discussion below.

Architecture analysis is rerun each time the specification is updated, to provide model checking of the architecture as the architecture itself is refined (early trade-off analysis of architectural styles and hardware/communication effects or changes during the development or lifecycle) and as software components are developed and refined. Properties reflect the attributes of hardware and software components, connections, and ports and along with language semantics are used in analyses. Properties can capture estimates, requirements, or component options. Estimates may proceed to final values based on measurement as development proceeds. The integration of property values (estimates to measured) can be compared to higher level requirements.

Multiple architecture analysis methods, such as schedulability, latency, safety, are selected and run on the system model as it is incrementally developed. Models can be high level, low fidelity abstractions for some analyses, or early in development. The specification is refined during development or as risks are discovered. Large models may be generated from system databases.

Analysis methods provide the cross checking needed to understand the effects of system change on

schedulability, latency, safety, utilization, fault tolerance etc. The analyses themselves can be incrementally added as new analysis tools become available by adding the appropriate properties throughout the system lifecycle. System of systems integrators can collect AADL specifications of lower level systems and analyze the higher level integrated system. Analysis methods and analysis tools will have to be selected consistent with the system architecture approach used. System developers will develop some special analysis for their own system implementation style and as a system engineering competitive advantage.

Software component source code is supplied by the user and can be generated from component generators (such as from Matlab/Simulink or Beacon), 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. However, with generation, prototypes can be rapidly developed to experiment with the system effects due to architecture changes or component changes that affect system performance.

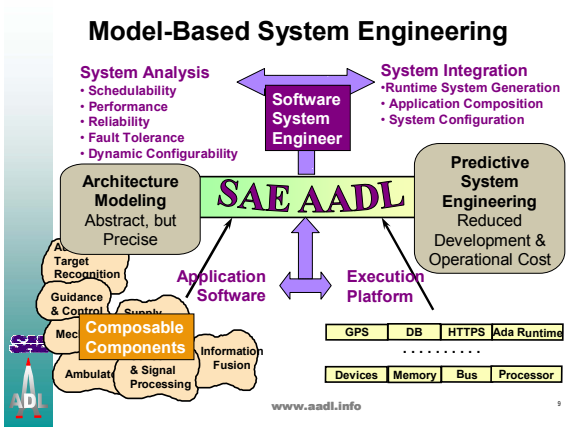


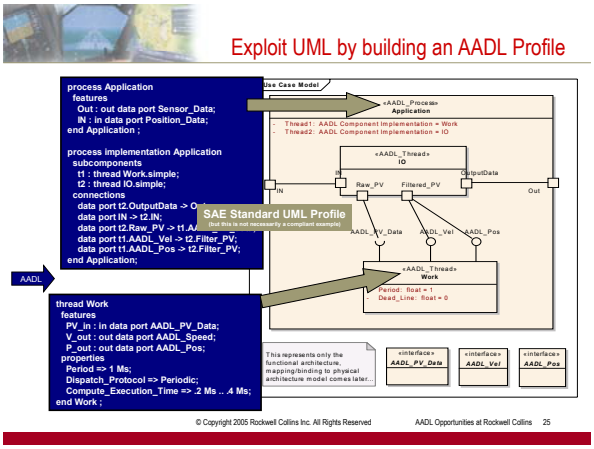
Figure 3 : Model Based System Engineering

3.2 Process Integration with UML, Simulink

The AADL UML profile will provide a means to integrate UML and AADL application development processes. The AADL profile will provide precise, component based semantics for capturing the computer hardware and software runtime architecture and to support its formal analysis. The figure below from Rockwell [39] provides an example of the mapping between UML and the AADL and illustrates their desire to use the AADL UML Profile.

The resulting model can then be analyzed by any AADL-based tool through the XML interchange format, or by tools that interface directly to the AADL profile in UML. UML tools could also support textual AADL output as a means of bridging to other tools. The TNI Europe toolset, STOOD [15], already provides a number of capabilities for integrated use of UML and AADL.

Stong interest has also been expressed for integration with Simulink from a number of AADL users. AADL components and semantics could be used as a bridge to Simulink or another component generator technology to guide component generation to the architectural specification. The AADL Programming Language annex [16] provides guidance for building/integrating AADL compliant systems in C and Ada. Several emerging AADL tools that have already demonstrated code integration/generation capability are listed under the tools section.



4. Tool Strategy and Progress

4.1 Open Source to Commercial

The SEI, via Dr. Peter Feiler, the principle author of the AADL standard, developed the Open Source AADL Tool Environment (OSATE) [17]. 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. Its available at http://www.aadl.info.

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 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 4 below.

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, prior to commercial tools. 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 [18] 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 as well is being developed to make it easy for UML tool vendors to support the AADL.

Commercial analysis tools may be integrated into 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, TNI Europe is extending their STOOD Hood/UML development environment to import, capture via AADL graphics, process and export AADL.

The AADL meta-model annex [19] includes the AADL Declarative Model and the AADL Instance model (see figure 4). The XML 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 processors. 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 in TOPCASED or GME [20].

Another important aspect of the standardized meta-model and XML schema is that it provides a database for analysis tool interaction that is standard. Tools can not only read from the XML 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, follow-on reliability, latency or safety analysis could be performed. Finally, from the system instance, the system could be automatically integrated with generation of glue code.

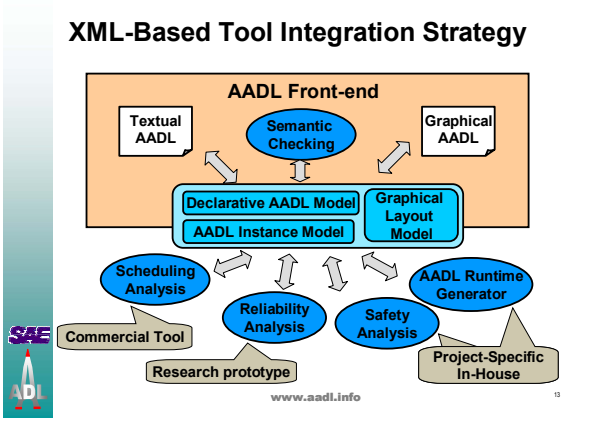


Figure 4 : Standardized and Simplified Tool Integration Supported

4.2 Additional Open Source Tools

There are a number of AADL open source toolsets either just becoming available or in the process of development. Here's a list.

- TOPCASED [10]
  - Airbus led, 22 partners participating, developing a meta-modeling framework, includes AADL toolset and Graphics, AADL XML, model transformation
- OCARINA [21]
  - ENST AADL toolset with graphics, middleware generation and integration to AADL specification of application on network distributed processors. Creates formal model of executive in AADL and Petri nets.
- MONTANA [22]

  - AADL to ACRS [23] (process algebra), formal analysis of concurrent resource utilization, scheduling
  - AADL to Charon [24], generation of control components and integration of hybrid control systems to AADL specification using Charon annex.
- GME [20]
  - Vanderbilt Univ, DARPA sponsored meta-modeling framework, AADL architecture specification and system security analysis being added.
- CHEDDAR [25]
  - ENST, advanced scheduling analysis toolset

5. 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 [26]. 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 [27]. ASSERT has modelled a dual redundant fault tolerant computer system using the annex [28,29].

6. AADL in Use

Some of the early adopter presentations on the AADL are highlighted in the slide in Figure 5. 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 [30]), 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 [8]), a presentation on an integrated UML and AADL process for the development of weapon system Plug & Play (PnP) architectures (General Dynamics [7]). Also included is a presentation on the modelling of a large modern aircraft system with architectural trade-off analysis, scheduling and safety analysis (Honeywell [27]) and a modelling and analysis of a modern helicopter architecture for system workload, partitioning and tuning of the switched network (Rockwell [31]). 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 [1]). 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 6 provides a list of Aircraft that Honeywell has modelled using MetaH/AADL [40]. This presentation also provides Honeywell's tool development strategy for AADL tools.

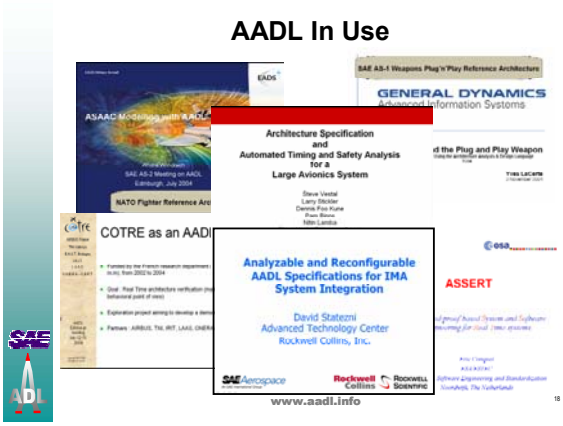


Figure 5: Demonstrations of AADL Capability

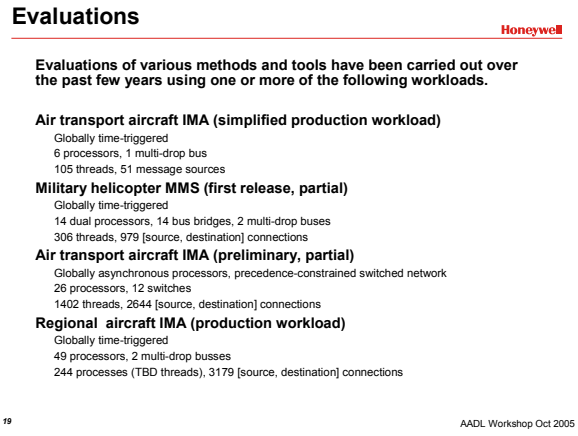


Figure 6: Honeywell MetaH/AADL Aircraft Modelling

7. AADL Transition Support

The Software Engineering Institute is providing transition support for the AADL in the US and Europe as part of their newly formed Predictable System Engineering Research 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 [32,33,34,35]. They have also developed (and are near publication) a Practitioner's Guide [36] on AADL and Control Systems Applications and a User's Guide [37] to AADL notation.

8. 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 benefit based on its standardized features including:

- Common modelling notation across organizations
- Single architecture model augmented with properties
- 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, including now AADL graphics. It is time to investigate the benefits of using the AADL on your systems in the aviation, space, and automotive domains.

9. Acknowledgement

The author would like to thank the AADL standardization committee for their diligent efforts and willingness to share through their presentations what they are doing with the AADL. Many of the papers referenced below have been developed by committee members.

10. References

1. Eric Conquet, " ASSERT – Automated proof-based System and Software Engineering for Real-Time systems », AADL Standardization Meeting, Edinburgh, July 2004
2. Pam Binns, Matt Englehart, Mike Jackson and Steve Vestal, "Domain Specific Software Architectures for Guidance, Navigation and Control," Honeywell Technology Center, Minneapolis, MN, International Journal of Software Engineering and Knowledge Engineering, Vol6, No. 2, 1996, pages 201-227.
3. David J. McConnell, Bruce Lewis and Lisa Gray, "Reengineering a Single Threaded Embedded Missile application onto a Parallel Processing Platform using MetaH," 5t Workshop on Parallel and Distributed Real Time Systems, 1996.
4. Jonathan W. Kruger, Steve Vestal, Bruce Lewis, "Fitting The Pieces Together: System/Software Analysis and Code Integration Using MetaH" Digital Avionics Systems Conference, 1998.
5. Douglas Gregory, "Aircraft, Launcher & Weapon Ineroperability – Review of Technical Findings », AADL Standardization Meeting, Edinburgh, July 2004
6. Andre Windisch, Herbert Schlatt, "AADL-Modelling of Plug&Play Weapon System Architecture", AADL Workshop, Paris, Oct 2004
7. Yves LaCerte, "AADL and MDA – Early Experience Applied to Aircraft-Weapon Integration", AADL Standardization Meeting, Seal Beach, Jan 2005
8. Patrick Farail, Pierre Dissaux, "COTRE a Software Design Workshop", DASIA 2002, May 2002.
9. Pierre Gaufillet, "COTRE as an AADL profile", AADL Standardization Meeting, Edinburgh, July 2004
10. Pierre Gaufillet, "TOPCASED – Toolkit In Open source for Critical Applications & SystEms Development", AADL Workshop, Paris, Oct 2005, http://www.topcased.org/
11. Society of Automotive Engineers (SAE) Avionics Systems Division (ASD) AS-2C Subcommittee. "Avionics Architecture Description Language Standard.", AS 5506, November 2004
12. Society of Automotive Engineers (SAE) Avionics Systems Division (ASD) AS-2C Subcommittee. "SAE Architecture Analysis & Design Language (AADL) Annex Volume 1 : Graphical AADL Notation, AADL Meta-Model and Interchange Formats, Language Compliance and Application Program Interface", proposed draft for publication, AS 5506 /1, Oct 2005.
13. Peter Feiler, Bruce Lewis, "The SAE AADL Standard : An Architecture Analysis & Design Language for Developing Embedded Real-Time Systems", World Computer Congress, ADL Workshop, Toulouse, August 2004
14. Joyce Tokar, Lutz Wrage, "The SAE Architecture Analysis and Design Language (AADL) Standard : A Basis for Architecture-Driven Embedded Systems

Engineering, International Workshop on Solutions for Automotive Software Architecture, Boston, Oct 2004

15. Pierre Dissaux, "Stood and the AADL", AADL Workshop, Paris, Oct. 2005, http://www.tni-world.com
16. Joyce Tokar, "SAE Architecture Analysis & Description Language – Programming Language Guidelines", AADL Workshop, Oct 2005
17. Peter Feiler, "Open Source AADL Tool Environment (OSATE)", AADL Workshop, Paris, Oct 2004
18. Mamoun Filali, "Cotre Annex HRT-HOOD embedding", AADL Workshop, Paris, Oct 2005
19. Peter Feiler, "AADL Meta Model & XML/XMI", AADL Workshop, Paris, Oct 2004
20. Matt Eby, Janos Mathe, Jan Werner, Chip Clifton, "Experimental Platform for Systems-Security Codesign", AADL Seminar, Dec 2005
21. Thomas Vergnaud, "The Ocarina Tool Suite", AADL Workshop, Paris, Oct 2005, http://ocarina.enst.fr
22. Oleg Sokolsky, "The Montana Toolset : OSATE Plugins for Analysis and Code Generation", AADL Workshop, Paris, Oct 2004
23. Duncan Clarke, Insup Lee, Hong-liang Xie, "VERSA : A Tool for the Specification and Analysis of Resource-Bound Real-Time Systems", University of Pennsylvania, May 1994
24. Rajeev Alur, Franjo Ivancic, Jesung Kim, Insup Lee, Oleg Sokolsky, "Generating Embedded Software from Hierarchical Hybrid Models", LCTES'03, San Diego, June 2003
25. F. Singhoff, J. Legrand, L. Nana, L. Marce, "Scheduling and Memory requirement analysis with AADL", Proceedings of the ACM SIGAda International Conference, Nov, 2005. http://beru.univ-brest.fr/~singhoff/cheddar/
26. Steve Vestal, "An Overview of the Architecture Analysis & Design Language (AADL) Error Model Annex", AADL Workshop, Paris, Oct 2005
27. Steve Vestal, "Automating Timing and Safety Analyses from Architecture Specifications", AADL Standardization Meeting, April 2005
28. Ana Rugina, Karama Kanoun, Mohamed Kaaniche, Jeremie Guiochet, "Dependability modelling of a fault tolerant duplex system using AADL and GSPNs", LAAS Research Report no. 05315, Draft of ASSERT Report, Sept 2005
29. Ana Rugina, "Dependability Modelling using AADL and the AADL Error Model Annex", AADL Workshop, Paris, Oct 2005
30. Andre Windisch, "ASAAC Modelling with AADL", AADL Standardization Meeting, Edinburgh, July 2004
31. David Statezni, "Analyzable and Reconfigurable AADL Specifications for IMA System Integration", SAE World Avation Conference, Reno, Oct 2004
32. Peter Feiler, David Statezni, "Multi-Fidelity Architecture Modeling", SEI internal report, Oct 2004
33. Peter Feiler, "Pattern-Based Analysis of an Embedded Real-Time System Architecture", World Computer Congress, ADL Workshop, Toulouse, August 2004
34. Peter Feiler, David Gluch, John Hudak, Bruce Lewis, "Embedded Systems Architecture Analysis Using SAE AADL", Technical Note, CMU/SEI-2004-TN-005, Software Engineering Institute, June 2004
35. Peter H. Feiler, Bruce Lewis, Steve Vestal, "Improving Predictability in Embedded Real-time Systems," Carnegie Mellon Software Engineering Institute, CMU/SEI-2000-SR-011, October 2000.
36. J. Hudak, "Developing AADL Models for Control Systems: A Practitioner's Guide", Technical Report CMU/SEI-2005-TR-022 December, 2005
37. D. Gluch, "User's Guide on the AADL Notation", Technical Report, March 2006
38. Society of Automotive Engineers (SAE) Avionics Systems Division (ASD) AS-2C Subcommittee "The SAE AADL Standard: A Language Summary" www.aadl.info/downloads/papers/AADLLanguageSummary.pdf
39. John Mettenburg, David Lempia, "AADL Opportunities at Rockwell Collins", AADL Seminar, Dec. 2005
40. Steve Vestal, "Methods and Tools for Embedded Distributed System Scheduling and Schedulability Analysis", AADL Workshop, Paris, Oct. 2005