AADLib, a library of reusable AADL models

ABSTRACT

The SAE Architecture Analysis and Design Language is now a well-established language for the description of critical embedded systems, but also cyber-physical ones. A wide range of analysis tools is already available, either as part of the OSATE tool chain, or separate ones.

A key missing elements of AADL is a set of reusable building blocks to help learning AADL concepts, but also experiment already existing tool chains on validated real-life examples.

In this paper, we present AADLib, a library of reusable model elements. AADLib is build on two pillars: 1/ a set of ready-to-use examples so that practitioners can learn more about the AADL language itself, but also experiment with existing tools. Each example comes with a full description of available analysis and expected results. This helps reducing the learning curve of the language. 2/ a set of reusable model elements that cover typical building blocks of critical systems: processors, networks, devices with a high level of fidelity so that the cost to start a new project is reduced.

AADLib is distributed under a Free/Open Source License to further disseminate the AADL language. As such, AADLib provides a convenient way to discover AADL concepts and tool chains, and learn about its features.

INTRODUCTION

The design and implementation of critical real-time embedded systems gather multiple domains, from low-level physics up to complex control of systems to implement a full function. Such complexity requires particular strategy to characterize each level of abstractions, and then integration to ensure the system under consideration is correctly built. The advent of Model-Based Engineering is often perceived as a silver bullet to achieve all these complex tasks: the system designer can master its design through proper model artifacts (blocks, connections, properties, etc.), virtual integration of system blocks, and analysis.

The agenda for these projects focus on the many required analysis: performance analysis (scheduling, network analysis, etc.), memory and processor (latency, jitter, issues with cache and pipelines), programming languages (simpler, smarter, notion of model of computation), formal methods (breaking limits in scalability of model checking, complexity of logic formula, notion of time, probability, etc.). Each analysis relies on a particular abstraction of the system: a model to be manipulated electronically for better efficiency.

Hence, model-based engineering (MBE) emerged as a convenient way to build models of systems to ease their analysis. Several tools have been developed, ECLIPSE being now the dominant platform, supporting UML and its companion profiles MARTE [1] and SysML [2], and the AADL language [3]. Proprietary tools like SCADE Studio or Matlab/Simulink are also available. Each tool supports different formalisms to express a system, and transformation engines to perform a wide range of analysis (such as behavioral, timing, safety) and eventually code generation.

The capability to define models and analyze them pave the way to virtual integration of subsystems and their analysis: a descriptive model of the system is built; the level of details of each block, and their interconnection is expressive enough to perform a complete analysis prior to actually build it. This allows for early trade-off analysis and detection of defects in the specifications, functional implementation or non-functional properties. Such analytic capabilities build upon existing model processing capabilities, typical analysis techniques but also require new innovative frameworks to address new level of complexities in design.

In this paper, we focus only on the AADL language, as we contributed to its latest updates and tool chains. We note that the interest by the industry in AADL is increasing, yet there is a lack of basic blocks on which one can either

1. Learn about AADL concepts and existing tool chains thanks to reusable examples
2. Have a library of reusable building blocks on which one can start a new project. These building blocks shall be complete enough to enable a wide range of

analysis.

In this paper, we introduce AADLib, a library of reusable model assets. AADLib is build on two pillars:

1. a set of ready-to-use examples so that practitioners can learn more about the AADL language itself, but also experiment with existing tools. Each example comes with a full description of available analysis and expected results. This helps reducing the learning curve of the language.
2. a set of reusable model elements that cover typical building blocks of critical systems: processors, networks, devices with a high level of fidelity so that the cost to start a new project is reduced.

AADLib is distributed under a Free/Open Source License to further disseminate the AADL language.

In the following, we introduce AADLv2 in next section. We then introduce AADLib and illustrate its features on several case studies.

OVERVIEW OF AADLV2

The "Architecture Analysis and Design Language" AADL is a textual and graphical language for model-based engineering of embedded real-time systems. It has been published as an SAE Standard AS-5506A [36]. AADL is used to design and analyze software and hardware architectures of embedded real-time systems.

The AADL allows for the description of both software and hardware parts of a system. It focuses on the definition of clear block interfaces, and separates the implementations from these interfaces. It can be expressed using both a graphical and a textual syntax. From the description of these blocks, one can build an assembly of blocks that represent the full system. To take into account the multiple ways to connect components, the AADL defines different connection patterns: subcomponent, connection, and binding.

An AADL model can incorporate non-architectural elements: embedded or real-time characteristics of the components (such as execution time, memory footprint), behavioral descriptions. Hence it is possible to use AADL as a back- bone to describe all the aspects of a system. Let us review all these elements:

An AADL description is made of *components*. The AADL standard defines software components (data, thread, thread group, subprogram, process) and execution plat- form components (memory, bus, processor, device, virtual processor, virtual bus) and hybrid components (system).

Each Component category describe well identified elements of the actual architecture, using the same vocabulary of system or software engineering:

- *Subprograms* model procedures like in C or Ada. *Threads* model the active part of an application (such as POSIX threads). AADL threads may have multiple operational modes. Each mode may describe a different behavior and property values for the thread. *Processes* are memory spaces that contain the *threads*. *Thread groups* are used to create a hierarchy among threads.

- *Processors* model microprocessors and a minimal operating system (mainly a scheduler). *Memories* model hard disks, RAMs, *buses* model all kinds of networks, wires, *devices* model sensors, …

- *Virtual bus* and *Virtual processor* models "virtual" hardware components. A virtual bus is a communication channel on top of a physical bus (e.g. TCP/IP over Ethernet); a virtual processor denotes a dedicated scheduling domain inside a processor (e.g. an ARINC653 partition running on a processor).

Unlike other components, *Systems* do not represent anything concrete; they combine building blocks to help structure the description as a set of nested components.

Packages add the notion of namespaces to help structuring the models. *Abstracts* model partially defined components, to be refined during the modeling process.

Component declarations have to be instantiated into sub-components of other components in order to model system architecture. At the top-level, a system contains all the component instances. Most components can have subcomponents, so that an AADL description is hierarchical. A complete AADL description must provide a top-most level system that will contain certain kind of components (*processor*, *process*, *bus*, *device*, *abstract* and *memory*), thus providing the root of the architecture tree. The architecture in itself is the instantiation of this system, which is called the *root system*.

The interface of a component is called *component type*. It provides *features* (e.g. communication ports). Components communicate one with another by *connecting* their *features*. To a given component type correspond zero or several implementations. Each of them describes the internals of the components: subcomponents, connections between those subcomponents, etc.

An implementation of a thread or a subprogram can specify *call sequences* to other subprograms, thus describing the execution flows in the architecture. Since there can be different implementations of a given component type, it is possible to select the actual components to put into the architecture, without having to change the other components, thus providing a convenient approach to configure applications.

The AADL defines the notion of *properties* that can be attached to most elements (components, connections, features, etc.). Properties are typed attributes that specify constraints or characteristics that apply to the elements of the architecture: clock frequency of a processor, execution time of a thread, bandwidth of a bus, . . . Some standard properties are defined, e.g. for timing aspects; but it is possible to define new properties for different analysis (e.g. to define particular security policies).

AADL is a language, with different representations. A *textual* representation provides a comprehensive view of all details of a system, and *graphical* if one want to hide some details, and allow for a quick navigation in multiple dimensions. In the following, we illustrate both notations. Let us note that AADL can also be expressed as a UML model following the MARTE profile [4].

The concepts behind AADL are those typical to the construction of embedded systems, following a component-based approach: blocks with clear interfaces and properties are defined, and compose to form the complete system. Besides, the language is defined by a companion standard document that documents legality rules for component assemblies, its static and execution semantics.

The following figure illustrates a complete space system, used as a demonstrator during the ASSERT project. It illustrates how software and hardware concerns can be separately developed and then combined in a complete model.

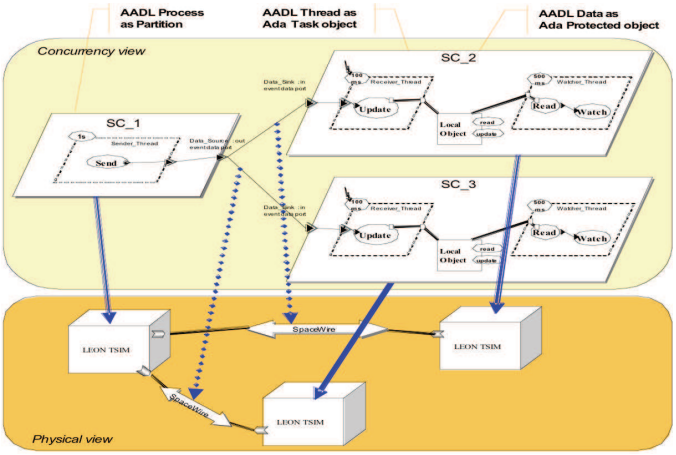

Figure 1: ASSERT MPC Case Study

EXISTING AADL TOOLS

AADL provides interesting features to model Critical Real-Time Embedded Systems, analyze them but also implement them. In this section, we show that there is currently a good coverage of support tools to assist designers. Actually, many tools provide support for AADL[1]:

- Modeling: TOPCASED [5], OSATE [6], and Stood [7] provide AADL modeling features for both textual and graphical variants;
- Model of computation and architectural patterns: AADLv2 annexes define patterns for supporting IMA architectures; other initiatives provides patterns for the Ravenscar computational model [8] or synchronous languages [9], [10];
- Scheduling analysis: the Fremont toolset [11] and Cheddar implement AADL performance analysis methods [12]. Gateways from AADL to the Cheddar and MAST tools are also available in TASTE;
- Dependability assessment; AADL provides an annex for modeling propagation of error, to be updated for AADLv2. Besides, connection with verification tools has been experimented for instance in the COMPASS project [13], the ADAPT toolset [14] and RT-Edge [15];
- Security: Patterns have been defined to model MILS security patterns [16], [17];
- Model optimization: optimization can occur across several dimensions: number of processors [18], use of communication buffers [19], allocation of threads to processors [20];
- Behavioral analysis: mapping to formal methods and associated model checkers have been defined for Petri Nets [21]; BIP [22], [23]; FIACRE [24]; RT-Maude [25];
- Performance analysis: performance of the system can be evaluated either at the level of generated source code [26], or from the interactions and I/Os in the system [27];
- Code generation: Ocarina implements Ada and C code generators for distributed systems [28], a mapping for RTSJ has been defined in [29]; AADS completes the range of language with hardware description language System-C [30]. Other initiatives exist to map AADL to synchronous languages like SIGNAL [31] or Lustre [32].

Several projects build on the foundations of these AADL tools to build integrated toolsets: the TASTE toolset driven by the European Space Agency [33]; the "System Architecture Virtual Integration" (SAVI) by the Aerospace Vehicle Systems Institute [34] an initiative gathering numerous key

[1] An updated list of supporting tools, projects and papers can be found on the official AADL web site http://www.aadl.info.

players from the aeronautics domain, and MASIW developed by the ISPRAS in Russia [35]. These integrated toolsets have to face many challenges, like the integration of additional modeling notations like SysML [37], or SCADE and Matlab/Simulink [36].

Hence, after more than 10 years of development around the AADL, and the seminal paper from [38], one can assert that AADL provides a complete toolbox for designing critical real-time embedded systems.

INTRODUCING AADLIB

From the existing projects, we note that there are severe differences. Besides, those can be blocking factors for the adoption of AADL in a full project:

- Some of them are research projects, at a very low TRL (Technical Readiness Level), in the 1-3 scale. This means those have been tested only on simple models, and may not scale to more complex ones. Also, these case studies are not always publically available
- Many of these projects were developed in the context of AADL1.0, and may require an update to handle the new features of AADLv2. The maintenance of these projects in not clear, and may not be performed by lack of support.

From these observations, we note that the state of AADLv2 tool support is unclear to an external user, interested in applying AADL concepts to his project. There is a need to have a clear view on the status of all these tools.

Furthermore, having many tools developed separately introduce an interesting interoperability issue: how to make sure that notionally equivalent models can be processed equally by tools that 1) provides the same kind of results (e.g. scheduling analysis) and 2) provides complementary results (e.g. safety and scheduling analysis of a system).

One contribution we present in this paper is a library of models elements to serve the community. This library proposes two complementary facets:

1. A library of reusable models to start a new project. Such library would provide a set of reusable patterns that could help starting new projects.
2. A library of example models, each of which presents either a particular modeling pattern or demonstrate how several analysis tools can be applied on this model.

The general objective of this library is to provide a central repository of AADL models geared towards the community. To be effective, this library should be easily integrated with existing AADL modeling environment, but also provide a large variety of examples.

To support these objectives, we initiated a project on the GitHub forge codenamed “AADLib” for AADL Library. This project proposes AADL models freely reusable, under a Free/Libre Software license, and is available at the following address: https://github.com/yoogx/AADLib.

This library can be imported in OSATE2 AADLv2 editor as a companion project, or used through the Ocarina suite of tools from the command line.

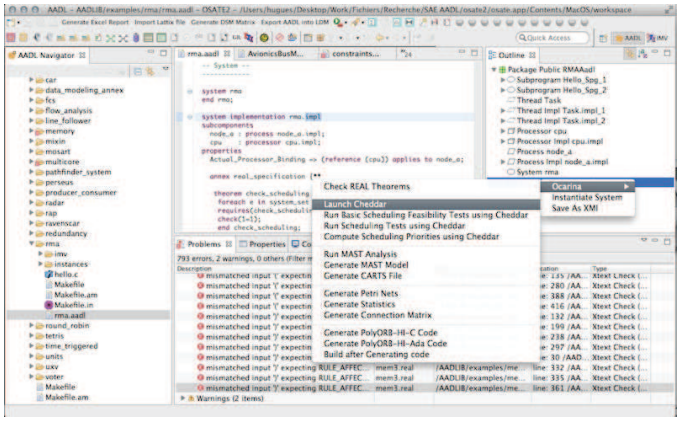

Figure 2 Ocarina OSATE2 Integration

In the next sections, we present the two facets of AADLib: 1) library of reusable blocks, 2) library of reference models.

AADLIB: REUSABLE BLOCKS

The first set of elements of AADLib aims at providing reusable building blocks. The requirements for such a library are many-fold:

- Provides a library of well-known building blocks: network interfaces, processors, devices, etc. These blocks are derived from existing elements, used widely in the industry (e.g. Ethernet, AFDX interfaces, Inertial Measurement Unit, processors, etc.),
- Complete the set of property sets proposed by AADLv2 core with advanced properties, describing key non-functional properties of a system,
- Propose modeling patterns to model and design both equipment-level subsystems and top-level systems.

AADLib proposes a full set of reusable blocks, proposing additions to the property sets defined in AADLv2 as well as reusable blocks

AADLIB property sets

AADLv2 supports a wide set of non-functional properties. Yet, to our surprise, some key properties are not present in the current standard, and could be of great help to provide a clear description of many blocks. AADLib provides additional properties. We list here the additional concerns modeled:

- **`processor_properties.aadl`**: this property set completes the properties applicable to processors with endianess, frequency, MIPS, FPU or multi-core concerns,
- **`bus_properties.aadl`**: adds bandwidth and channel type (duplex, half-duplex) considerations,
- **`data_sheet.aadl`**: connects AADL model entities to data sheets or bill of materials for physical implementation,
- **`electricity_properties.aadl`**: covers energy converters and electric units. This is useful for characterizing devices or processor consumptions.
- **`physical_properties.aadl:`** adds other units for power, mass, etc.
- **`memory_segments.aadl`**: extend the description of memory components with fine-grained definition of segment or page descriptors.

These properties help providing a full description of a system, it is used intensively to model the blocks forming the library of reusable AADL elements provided by AADLib.

AADLib reusable model elements

In addition to extended property sets, AADLib proposes a set of building blocks. These blocks provide a valuable asset to start new models. The library is built following AADL model hierarchy of elements:

- Processors: various ARM, AVR, PowerPC, SPARC, x86 processors are available, with endianess, frequency, ports modeled;
- Buses: typical network interfaces are modeled, covering AFDX, ARINC429, CAN, Ethernet, I2C, MIL-STD 1553, PCI, SpaceWire, UART, USB, with known limits in bandwidth, packet size, etc.,
- Miscellaneous devices: battery, GPS, accelerometers, inertial measurement units, etc. Those are modeled after components we use for teaching real-time or embedded systems in our classes at ISAE,
- Full systems, modeled after known reference design: Arduino, Aeroflex Gaisler boards, Wind River SBCs.

The library is organized so that each component type and associated implementation are in a separate AADL package so as to ease inclusion in large scale projects.

AADLIB: EXAMPLE MODELS

As we mentioned earlier, the diversity in AADL tool support and analysis capabilities make it difficult to learn AADL concepts and apply them using well-chosen tools. To cope with this issue, we propose a set of example models, all compatible with OSATE2 and Ocarina, and for each model we list compatible analysis tools.

For now, we focused mainly on analysis tools available through Ocarina, and for compatibility with OSATE2, we made sure this tool accepted all AADLib models. Furthermore, Ocarina features are also available through OSATE2 thanks to a dedicated plug-in.

Thanks to Ocarina functionalities, the following AADL processing capabilities are available:

- Scheduling analysis, testing with either Cheddar or MAST. Each tool having its own set of feasibility tests, it is important to offer diversity,
- Model-checking, targeting either colored Petri Nets (through CPN-AMI), or Timed Petri Nets (through TINA), and associated tools for the verification of temporal logic predicates,
- Code generation for C or Ada AADL runtimes, allowing for rapid prototyping of an AADL system for deployment on various RTOS or native targets,
- Worst-Case Execution-Time (WCET) analysis, based on the previous code generator,
- Constraint checking, using the REAL constraint language attached to validate a model conforms to a set of architectural restrictions (e.g. MILS, ARINC653, etc.).

We organized the library to either demonstrate a basic capability of the AADL (e.g. scheduling) or how to build larger systems. For each model, a full description provides information on what can be achieved. We list some of the available models, to give an overview of the diversity of concerns addressed:

- aocs: model of an Attitude Orbital Control System, derived from an ESA technical report,
- ardupilot: models the ArduPilot UAV platform, adapted from the POK project
- arinc653_annex: models from the ARINC653 annex document for AADLv2
- asl: work in progress REAL theorems in preparation for the ASL annex as part of SAE AS2/C committee work
- fcs: models a naive Flight Control Systems. Only the architecture is developed, for analyzis with Cheddar.
- line_follower: a line follower robot for the Arduino platform, using some parts available from SparkFun Electronics.
- memory: this examples demonstrates how to define logical and hardware memory layout, and how to ensure they match.
- mixin: this example demonstrates how to support multiple inheritance in AADLv2, using the “mixin” pattern defined in many object-oriented languages like Ada.
- multicore: provides one solution for modeling multicore systems in AADLv2, and performing code generation using Ocarina.
- pathfinder_system: models the well known pathfinder probe, and illustrates its priority inversion problem using Cheddar.

- radar: a naive model of a radar system
- rap: Ravenscar Avionics Platform, written by Olivier Gilles during his PhD. This models builds upon the Generic Avionics Platform from SEI.
- ravenscar: case study issued from the ``Guide for the use of the Ada Ravenscar Profile in high integrity systems'' written by Alan Burns, Brian Dobbing and Tullio Vardanega. This model has been translated to AADLv2, and extended to include REAL theorems to check Ravenscar patterns.
- rma: two tasks with different period on the same node, can be checked by Cheddar, or can generate code for either PolyORB-HI/C or Ada.
- time_triggered: shows how to implement a time-triggered architecture using delayed connections.
- uxv: models a series of UAV and UGV from ISAE DMIA lab. Using REAL, we can evaluate the energy consumption of the system.

Through this extensive set of examples, one can learn most of AADLv2 concepts and apply them directly in a comprehensive toolset. The figure below reproduces the GANTT chart generated from a simulation of the PathFinder system modeled in AADLv2.

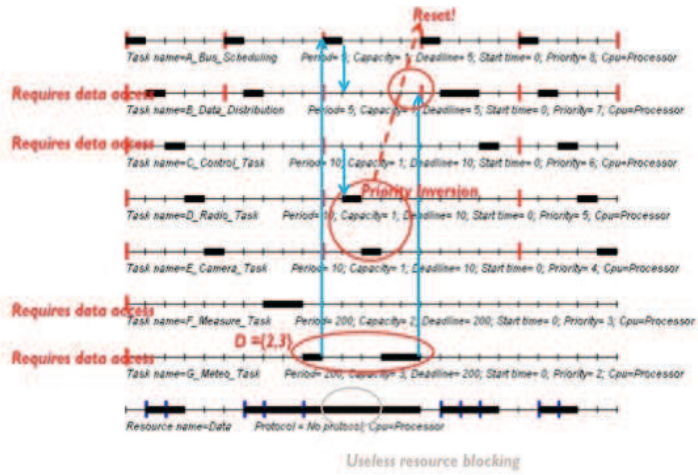

Figure 3 Mars Pathfinder analysis from AADLv2 model

CONCLUSION

AADLv2 is now a well-established notation for modeling critical, real-time embedded systems in all their complexity. We presented the language and associated tools. We also underlined the lack of a common library of reusable model elements and examples.

To support these objectives, we initiated a project on the GitHub forge codenamed "AADLib" for AADL Library. This project proposes AADL models freely reusable, under a Free/Libre Software license, and is available at the following address: https://github.com/yoogx/AADLib.

This library of models builds on top of many years of expertise in AADL we developed as part of the ASSERT and TASTE projects, and interaction within the AADL community.

As of today, AADLib provides more than 100 reusable components, and 25 example models. Furthermore, AADLib is mostly focused on Ocarina tool support.

Future work will concentrate on two directions

- Extend the tools that can operate on AADLib models, e.g. fault analysis tools being developed by the SEI;
- Extend the library of models to new family of systems.

AADLib being open source, developed in an open way, we welcome contributions and comments with a strong interest!

REFERENCES

1. OMG. *A UML Profile for MARTE, version 1.1*. OMG Document Number: formal/2011-06-02, June 2011.
2. OMG. *OMG Systems Modeling Language (OMG SysML)*. OMG Document Number: formal/2010-06-01, June 2010.
3. SAE. Architecture Analysis and Design Language (AADL) AS-5506A. Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 2.0, January 2009.
4. Madeleine Faugere, Thimothee Bourbeau, Robert de Simone, and Sebastien Gerard. Marte: Also an UML profile for modeling AADL applications. *Engineering of Complex Computer Systems, IEEE International Conference on*, 0:359–364, 2007.
5. P. Farail, P. Gaufillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel. TOPCASED : An Open Source Development Environment for Embedded Systems. *Chapter 11, From MDD Concepts to Experiments and Illus- trations, ISTE Editor*, pages 195–207, September 2006.
6. SEI. OSATE : An extensible Source AADL Tool Environ- ment. *SEI AADL Team technical Report*, December 2004.
7. P. Dissaux. Using the AADL for mission critical software development. *2nd European Congress ERTS, EMBEDDED REAL TIME SOFTWARE Toulouse*, January 2004.
8. Olivier Gilles and Jerome Hugues. Expressing and enforcing user-defined constraints of AADL models. In *Proceedings of the 5th UML& AADL Workshop (UML&AADL 2010)*, pages 337–342, University of Oxford, UK, March 2010.
9. Zhibin Yang, Kai Hu, Jean-Paul Bodeveix, Lei Pi, Dianfu Ma, and Jean-Pierre Talpin. Two formal semantics of a subset of the aadl. In Perseil et al. [31], pages 344–349.

10. Peter Csaba Ölveczky, Artur Boronat, and José Meseguer. Formal semantics and analysis of behavioral aadl models in real-time maude. In John Hatcliff and Elena Zucca, editors, *FMOODS/FORTE*, volume 6117 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2010.
11. O. Sokolsky, I. Lee, and D. Clark. Schedulability Analysis of AADL models . International Parallel and Distributed Processing Symposium, IPDPS 2006,, April 2006.
12. F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand. Investigating the usability of real-time scheduling theory with the Cheddar project. *Journal of Real-Time Systems, Springer Verlag*, 43(3):259–295, November 2009.
13. Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. The com- pass approach: Correctness, modelling and performability of aerospace systems. In *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security*, SAFECOMP '09, pages 173–186, Berlin, Heidelberg, 2009. Springer-Verlag.
14. Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaâniche. The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. In *EDCC*, pages 85–90. IEEE Computer Society, 2008.
15. Myron Hecht, Alexander Lam, and Chris Vogl. A tool set for integrated software and hardware dependability analysis using the architecture analysis and design language (aadl) and error model annex. In Perseil et al. [31], pages 361–366.
16. Julien Delange, Laurent Pautet, and Fabrice Kordon. Design, Verification and Implementation of MILS systems. In *Proceedings of the 21th International Symposium on Rapid System Prototyping*, pages 1–8, Fairfax, June 2010. IEEE Computer Society. MoVe INT LIP6.
17. Jorgen Hansson, Bruce Lewis, Jerome Hugues, Lutz Wrage, Peter Feiler, and John Morley. Model-Based Verification of Security and Non-Functional Behavior using AADL. *IEEE Security & Privacy*, 8(1):43–49, January 2010.
18. Olivier Gilles and Jérôme Hugues. Towards Model- based optimisations of Real-Time systems, an application with the AADL. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009)*, Pekin, Chine, August 2009.
19. Peter H. Feiler. Efficient embedded runtime systems through port communication optimization. In *ICECCS*, pages 294– 300. IEEE Computer Society, 2008.
20. Dionisio de Niz and Peter H. Feiler. On resource allocation in architectural models. In *ISORC*, pages 291–297. IEEE Computer Society, 2008.
21. Xavier Renault, Fabrice Kordon, and Jerome Hugues. Adapt- ing models to model checkers, a case study : Analysing AADL using Time or Colored Petri Nets. In *IEEE/IFIP 20th International Sypmosium on Rapid System Prototyping* , Paris, France, June 2009.
22. Lei Pi, Jean-Paul Bodeveix, and Mamoun Filali. Modeling aadl data communication with bip. In Fabrice Kordon and Yvon Kermarrec, editors, *Ada-Europe*, volume 5570 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2009.
23. Mohamed Yassin Chkouri, Anne Robert, Marius Bozga, and Joseph Sifakis. Translating aadl into bip - application to the verification of real-time systems. In Michel R. V. Chaudron, editor, *MoDELS Workshops*, volume 5421 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 2008.
24. Bernard Berthomieu, Hubert Garavel, Frédéric Lang, and François Vernadat. Verifying dynamic properties of indus- trial critical systems using topcased/fiacre. *ERCIM News*, 2008(75), 2008.
25. Peter Csaba Ölveczky, Artur Boronat, and José Meseguer. Formal semantics and analysis of behavioral aadl models in real-time maude. In John Hatcliff and Elena Zucca, editors, *FMOODS/FORTE*, volume 6117 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2010.
26. Olivier Gilles and Jerome Hugues. Applying WCET analysis at architectural level. In *Worst-Case Execution Time (WCET'08)*, pages 113–122, Prague, Czech Republic, July 2008.
27. Min-Young Nam, Rodolfo Pellizzoni, Lui Sha, and Richard M. Bradford. Asiist: Application specific i/o integration support tool for real-time bus architecture designs. In *ICECCS*, pages 11–22. IEEE Computer Society, 2009.
28. Gilles Lasnier, Bechir Zalila, Laurent Pautet, and Jérôme Hugues. OCARINA: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Reliable Software Technologies'09 - Ada Europe*, volume LNCS, pages 237–250, Brest, France, June 2009.
29. Jean-Paul Bodeveix, Raphaël Cavallero, David Chemouil, Mamoun Filali, and Jean-François Rolland. A mapping from aadl to java-rtsj. In Gregory Bollella, editor, *JTRES*, ACM International Conference Proceeding Series, pages 165–174. ACM, 2007.
30. Roberto Varona-Gomez and Eugenio Villar. Aadl simulation and performance analysis in systemc. In *Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS '09, pages 323–328, Washington, DC, USA, 2009. IEEE Computer Society.
31. Yue Ma, Huafeng Yu, Thierry Gautier, Jean-Pierre Talpin, Loïc Besnard, and Paul Le Guernic. System Synthesis from AADL using Polychrony. In *Electronic System Level Synthesis Conference*, San Diego, California, États-Unis, June 2011.
32. Erwan Jahier, Nicolas Halbwachs, Pascal Raymond, Xavier Nicollin, and David Lesens. Virtual Execution of AADL Models via a Translation into Synchronous Programs. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 134 – 143, Salzburg, Autriche, 2007. ASSERT.
33. Eric Conquet, Maxime Perrotin, Pierre Dissaux, Thanassis Tsiodras, and Jerome Hugues. The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software . In

Proceedings of Embedded Real Time Software and Systems 2010, Toulouse, France, May 2010.

34. Peter Feiler, Joergen Hansson, and Dioniio de Niz. System Architecture Virtual Integration: An Industrial Case Study. Technical report, Software Engineering Institue, Carnegie Mellon University, 2009.
35. Alexey V. Khoroshilov, Igor Koverninskiy, Alexandre Pe- trenko, and Alexander Ugnenko. Integrating aadl-based tool chain into existing industrial processes. In Perseil et al. [31], pages 367–371.
36. Julien Delange, Jerome Hugues, Laurent Pautet, and Diosisi odeNiz. AMDE-basedProcessfortheDesign,Implementa- tion and Validation of Safety Critical Systems. In *Proceedings of the 5th UML& AADL Workshop (UML&AADL 2010)*, pages 319–324, University of Oxford, UK, March 2010.
37. Razieh Behjati, Tao Yue, Shiva Nejati, Lionel Briand, and Bran Selic. An aadl-based sysml profile for architecture level systems engineering: Approach, metamodels, and ex- periments. Technical Report 2011-03, Simula Research Lab- oratory, 2011.
38. Robert Allen,, Steve Vestal, Dennis Cornhill, and Bruce Lewis. Using an architecture description language for quantitative analysis of real-time systems. In *Proceedings of the 3rd international workshop on Software and performance*, WOSP ’02, pages 203–210, New York, NY, USA, 2002. ACM.