To cite this document: Hugues, Jérôme and Singhoff, Frank *AADLv2, an Architecture Description Language for the Analysis and Generation of Embedded Systems.* (2013) In: ESWeek'2013 - Embedded Systems Week, 29 September 2013 (Montréal, Canada). (Unpublished)

AADLv2, an Architecture Description Language for the Analysis and Generation of Embedded Systems

Jérôme Hugues
Université de Toulouse, ISAE
10, Avenue E. Belin 31055 Toulouse Cedex 4, France
Email: jerome.hugues@isae.fr

Frank Singhoff
Lab-STICC UMR CNRS 6485
Université de Bretagne Occidentale – UEB
20, avenue le Gorgeu
29238 Brest Cedex 3, France
Email: singhoff@univ-brest.fr

***Abstract*—The Architecture Analysis and Design Language (AADL) is an SAE International Standard dedicated to the precise modeling of complex embedded systems, covering both hardware and software concerns. Its definition relies on a precise set of concepts inherited from industry and academics best practice: clear separation of concerns among layers, rich set of properties to document system metrics and support for many kind of analysis: scheduling, safety and reliability, performance, but also code generation.**

In this tutorial, we provide an overview of AADLv2 and illustrate how several analyses can be combined on an illustrative example: a UAV platform.

I. Overview of the AADL

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-5506B [7]. AADL is used to design and analyze the architecture of embedded real-time systems.

AADL allows for the description of both software and hardware parts of a system. It focuses on the definition of block interfaces, and separates the implementations from these interfaces. It can be expressed using both a graphical or a textual syntax. From the description of these blocks, one can assemble blocks to represent the full system.

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

An AADL description is made of *components*. The AADL standard defines software components (`data`, `thread`, `thread group`, `subprogram`, `process`) and execution platform 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 behaviour 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 micro-processors and a minimal operating system (mainly a scheduler). *Memories* model hard disks, RAMs, *buses* model all kinds of networks.
- *Virtual bus* and *Virtual processor* models logical point of view of hardware components. A virtual bus is a communication channel on top of a physical bus; 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 subcomponents of other components in order to model an 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*. A component type can have several implementations. They describe the internals of the components: subcomponents, connections between those subcomponents, ...

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,

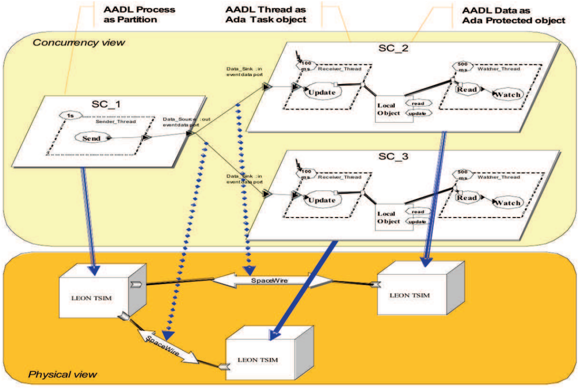

Fig. 1. IST-ASSERT demonstrator

…). 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 security policies).

AADL is a language, with different representations. A *textual* representation provides a comprehensive view of all details of a system. A *graphical* representation also exists if one want to hide some details and to quick navigate in multiple dimensions of the architecture model. In the following, we illustrate both notations. Let us note that AADL can also be expressed as a UML model following the MARTE profile [3].

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 figure 1 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.

II. AADL FOR EMBEDDED SYSTEMS

AADL provides interesting features to model embedded systems, analyze them but also implement them. In this section, we review some existing tools[1]:

- *Modeling:* OSATE [8], and Stood [2] provide AADL modeling features for both textual and graphical variants;
- *Model of computation and architectural patterns:* AADLv2 annexes define patterns for supporting IMA architectures, the Ravenscar [5] or Synchronous computational models;
- *Scheduling analysis:* using Cheddar [9] or its industrial version AADLInspector;
- *Dependability:* using Error modeling annex of AADLv2;
- *Code generation:* Ocarina implements Ada and C code generators for distributed systems [6].

Many integrated industry-driven projects rely on these tools : the TASTE toolset driven by the European Space Agency [1] or the "System Architecture Virtual Integration" (SAVI) by the Aerospace Vehicle Systems Institute [4], an initiative gathering numerous key players from the aeronautics domain.

III. ABOUT THE TUTORIAL

The tutorial illustrates the two key dimensions of AADL: 1) a modeling process following a system engineering approach, 2) connection with various analysis down and up the traditional engineering V-cycle. The tutorial will cover both language and state-of-the- art tools: OSATE2, Cheddar and Ocarina and connections with other tools like Simulink or OpenFTA.

We illustrate[2] how to merge various modeling and analysis concerns using AADL: validation of mission-level objectives, high-level system validation, verification of schedulability or reliability and code generation.

ACKNOWLEDGMENTS

The author thanks the members of the AS2-C committee on the AADL, and members of European Space Agency and Ellidiss Technologies for their valuable support of AADL-related activities.

REFERENCES

[1] 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.

[2] P. Dissaux. Using the AADL for mission critical software development. *2nd European Congress ERTS, EMBEDDED REAL TIME SOFTWARE Toulouse*, January 2004.

[3] 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.

[4] 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.

[5] 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.

[6] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and JÃ©rÃ´me 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.

[7] SAE. Architecture Analysis and Design Language (AADL) AS-5506B. Technical report, September 2012.

[8] SEI. OSATE : An extensible Source AADL Tool Environment. *SEI AADL Team technical Report*, December 2004.

[9] 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.

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

[2] All models used in the tutorial are available on the authors web pages.