CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Open Archive Toulouse Archive Ouverte

Open Archive TOULOUSE Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 14207

To cite this version : Hugues, Jérôme and Delange, Julien AADLv2, a Domain Specific Language for the Modeling, the Analysis and the Generation of Real-Time Embedded Systems. (2015)
In: Models Conference 2015, 27 September 2015 - 2 October 2015 (Ottawa, Canada)

Any correspondance concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

AADLv2, a Domain Specific Language for the Modeling, the Analysis and the Generation of Real-Time Embedded Systems

Julien Delange
Carnegie Mellon Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612, USA
Email:jdelange@sei.cmu.edu

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

I. Title

AADLv2, a Domain Specific Language for the Modeling, the Analysis and the Generation of Real-Time Embedded Systems

II. Presenters bio

- Julien Delange (Software Engineering Institute, Pittsburgh, USA) is a researcher at the Carnegie Mellon University (CMU) Software Engineering Instute (SEI). Julien got a PhD (2010) in Computer Science that focuses on the design of safe and secure embedded systems. Prior working at the SEI, he worked at the European Space Agency and was involved in many aerospace-related projects for using model-based engineerins approaches to improve the development of safety-critical systems.
- Jérôme Hugues (ISAE/DMIA, Toulouse, France) is associate professor at the Department of Mathematics, Computer Science, and Control of the Institute for Space and Aeronautics Engineering (ISAE). He holds a PhD (2005) and an engineering degree (2002) from Telecom ParisTech. His research interests focus on design of software-based real-time and embedded systems and tools to support it. He is a member of the SAE AS-2C committee working on the AADL; and is involved in the Ocarina and TASTE projects, two flagships AADL projects.

III. Abstract

The Architecture Analysis and Design Language (AADL) is an SAE International Standard dedicated to the precise modeling of complex real-time 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 radar platform. This tutorial, we also present Model-based engineering process allowed by AADL to both verify and implement automatically a real-time embedded systems.

IV. Keywords

Model-based engineering; AADL; semantics; safety analysis; code generation

V. Proposed length of the lecture

3 hours

VI. Level of the tutorial

Advanced

VII. Target audience and any pre-requisite background required by attendees

This tutorial requires basic knowledged on Model Driven Engineering. As both AADL and real-time verification methods will be introduced, this tutorial does not assume skills on them.

VIII. Description of the tutorial

A. 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 [1]. 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 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 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). 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.

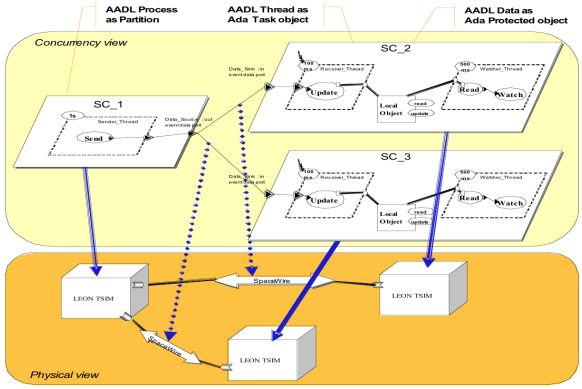

Fig. 1. IST-ASSERT demonstrator

The AADL defines the notion of *properties* that can be attached to most elements (components, connections, features, ...). 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 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 [2].

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, as a demonstrator during the ASSERT project. It illustrates software and hardware concerns can be separately developed and then combined in a complete model.

As we mentioned earlier, AADL, or other like MARTE or EAST-ADL provides similar constructs, and are conceptually really closed as underlined in [3].

B. 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 [4], and Stood [5] provide AADL modeling features for both textual and graphical variants;

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

- *Model of computation and architectural patterns:* AADLv2 annexes define patterns for supporting IMA architectures, the Ravenscar [6] or Synchronous computational models;
- *Safety analysis:* using OSATE plug-ins mapping AADL concepts onto fault trees, and supporting FHA and FMEA analysis;
- *Code generation:* Ocarina implements Ada and C code generators for distributed systems [7].

Many integrated industry-driven projects rely on these tools : the TASTE toolset driven by the European Space Agency [8] or the "System Architecture Virtual Integration" (SAVI) by the Aerospace Vehicle Systems Institute [9], an initiative gathering numerous key players from the aeronautics domain.

C. 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 SCADE.

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

IX. Intended outline

As the goal of this tutorial is to introduce model-based analysis embedded systems using the AADLv2 Architecture Description Language, the outline of the tutorial will be as follow:

- Part 1: introduction to AADLv2 core (about 60 minutes). We will present the syntax and semantics of the AADL.
- Part 2: introducing a case study (about 30 minutes). We present an avionics case study to illustrate the use of AADL.
- Part 3: safety analysis (about 60 minutes). We will introduce the requirements of safety analysis and how safety documentation can be generated from AADL.
- Part 4 : code generation (about 60 minutes). Finally, we will present how to generate code from an AADL model for avionics (ARINC653) architectures and how it can be run.

X. Novelty of the tutorial

This tutorial has been given by the speakers at the ESWeek 2013, HILT 2014 and MODELS 2014 conferences. The attendees of these tutorials were interested in, but not expert on real-time embedded systems. J. Hugues also delivered a short version of this tutorial, in French, at Ecole dété Temps Réel, the French summer school on real-time systems in 2009.

This occurrence would be adapted and enhanced for this new occurence to include latest developmeent on the AAD standard, and its associated toolset. We plan to provide a stronger focus on usage of the OSATE AADL toolset.

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

XI. Sample slides

Slides from the previous tutorial are available at the following URL: http://www.openaadl.org/post/2014/09/28/models/

References

[1] SAE: Architecture Analysis and Design Language (AADL) AS-5506B. Technical report (2012)

[2] Faugere, M., Bourbeau, T., de Simone, R., Gerard, S.: Marte: Also an uml profile for modeling aadl applications. Engineering of Complex Computer Systems, IEEE International Conference on **0** (2007) 359–364

[3] Johnsen, A., Lundqvist, K.: Developing dependable software-intensive systems: Aadl vs. east-adl. In Romanovsky, A., Vardanega, T., eds.: Ada-Europe 2011, Springer-Verlag (2011) 103–117

[4] SEI: OSATE : An extensible Source AADL Tool Environment. SEI AADL Team technical Report (2004)

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

[6] Gilles, O., Hugues, J.: Expressing and enforcing user-defined constraints of AADL models. In: Proceedings of the 5th UML& AADL Workshop (UML&AADL 2010), University of Oxford, UK (2010) 337–342

[7] Lasnier, G., Zalila, B., Pautet, L., Hugues, J.: OCARINA: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In: Reliable Software Technologies'09 - Ada Europe. Volume LNCS., Brest, France (2009) 237–250

[8] Conquet, E., Perrotin, M., Dissaux, P., Tsiodras, T., Hugues, J.: 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 (2010)

[9] Feiler, P., Hansson, J., de Niz, D.: System Architecture Virtual Integration: An Industrial Case Study. Technical report, Software Engineering Institue, Carnegie Mellon University (2009)