Comparison of the AADL and Event-B Model-Based Tool Chains for Designing Embedded Systems

Christophe Ponsard and Renaud De Landtsheer

CETIC Research Centre, Charleroi (Belgium) - {cp, rdl}@cetic.be

Abstract. This paper compares two model-based tool chains supporting the design of embedded systems. The first one relies on an architecture description language called AADL (Architecture Analysis and Design Language) specially targeting embedded system. The second one is based on a generic notation for system modelling called Event-B which is being deployed in the design of embedded systems, e.g. in the transportation sector. A number of tools supporting similar activities are highlighted as well as the impact of the characteristics of the respective underlying languages on the global design activity.

1 Introduction

The development of model-driven engineering (MDE) is opening new perspectives in the development of computer-based systems. MDE makes it possible to define more precisely a number of design artifacts, reason on them at an early stage, and interconnect them more tightly. It also supports the use of generative or formally verified approaches. An efficient support of Computer Assisted Software Engineering (CASE) tool is known to be critical in any development process [9]. This is even more blatant for MDE were fully integrated tool chains must support the chain of models and the relations between them.

MDE also applies to embedded systems, where specific issues are to be dealt with, such as the design of the hardware/software boundary, or and also a number of well-categorized non-functional requirements such as performance (real-time, worst-case time, memory/CPU consumption) and RAMS (Reliability, Availability, Maintainability and Safety) [10].

The purpose of this paper is to compare two tool chains relying on two different modelling approaches.

- the first tool chain is based on a domain-specific language for describing architecture of embedded systems: the **AADL** language (Architecture Analysis and Design Language) which originates from the industry (automotive/aeronautics) [17]. It is supported by a number of tools that have been integrated on the TOPCASED environment[21].
- the second tool chain is based on a generic system modelling notation called **Event-B** which is developed by a mixed consortium of academics and industrial partner and industrial partners. It shares the same mathematical grounds as the B language and is supported by the RODIN toolset [7].

This work is more directed towards identifying fundamental nature of some benefits and limitations rather than implementation related issues. To ease this, we selected solutions with similar implementations relying on the Eclipse platform and related technologies. There is also a strong research dimension in both.

This paper is structured as follows. Section 1 and 2 respectively presents the AADL and Event-B languages and related tool chains. A comparative discussion is then carried out in section 3 and is followed by a conclusion in section 4.

2 Tool Chain based on AADL

2.1 The AADL language

AADL (Architecture and Analysis Design Language) is a standard language for describing software and hardware components of a system and how they map to processors on the execution platform [17]. At structural level, an AADL model is mainly composed of:

- *components*: also called *elements*. They are divided into hardware, software and composite categories (see figure 1). Each component is described at two levels: the *type level* represents the functional interface of the component. The *implementation level* describes the contents of the component and is expressed as subcomponents and connectors. Properties can be associated to components and enable a characterization of the component (e.g period, deadline for real-time).
- *connectors*: describe the data and control flows between components. They rely on the use of ports and connections. A *port* is an entry or exit point in a component, where data, events, or associated data and events may transit. A *connection* enables the link between two ports, either the ports of two subcomponents, or the port of a subcomponent and the port of its owner component.

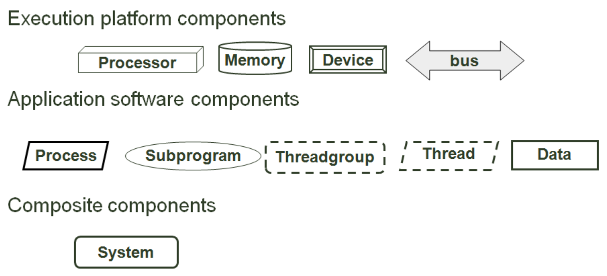

Fig. 1. Graphical Notations for AADL

The behavioural level was not precisely defined in the first release of AADL in 1994 but was defined in a recent extension (behavioural annex) [8].

The above shows that the language is rather grown in a bottom-up approach with progressive extensions and more semantically precise definitions. In addition to the graphical syntax shown in figure 1, AADL has also a fully textual syntax.

2.2 Tool Chain Description.

Over the time, a number of tools emerged to support activities related to the AADL model. The SPICES ITEA project produced a first fully integrated prototype of a complete tool chain, based on the TOPCASED platform [20, 21]. These tools are depicted in figure 2 and the most representative are described below (model editors will not be detailed).

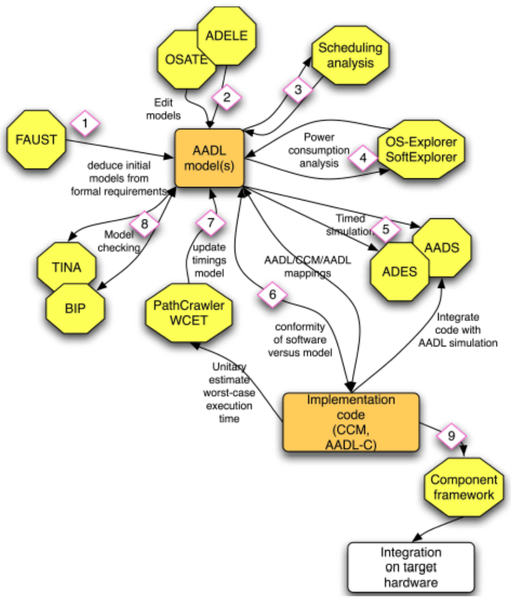
ADELE
OSATE
Scheduling analysis
Edit models
2
3
1
FAUST
AADL model(s)
Power consumption analysis
4
OS-Explorer SoftExplorer
deduce initial models from formal requirements
8
Model checking
7
update timings model
Timed simulation
5
AADS
ADES
TINA
BIP
AADL/CCM/AADL mappings
PathCrawler WCET
6
conformity of software versus model
Integrate code with AADL simulation
Unitary estimate worst-case execution time
Implementation code (CCM, AADL-C)
9
Component framework
Integration on target hardware

Fig. 2. SPICES AADL Tool Chain

Requirements Engineering. *FAUST* [16] is a model-based requirements engineering toolbox supporting the formal layer of the KAOS goal-oriented language [22]. It enables the capture and structure of requirements in goal refinement graphs and their assignment to responsible software, hardware or environmental agents. *Goals* can be formally specified using real-time linear temporal logic (LTL) [13]. To transition from requirements to AADL, a mapping supporting both the structural and behavioural dimensions was defined and partly implemented [15].

Model Checking. *TINA* is a toolbox allowing to model, simulate, perform model checking of timed systems[4]. In particular, it includes analysis of the scheduling using model checking techniques. It includes a pivot language, called *Fiacre* enabling interface several modelling languages as input. TINA can check the model against LTL properties identified at the requirements level using Petri Nets.

Power Consumption. SoftExplorer, now called CAT (Consumption Analysis Toolbox) [18] allows the estimation of system-level power and energy consumption from models. Methods and models have been developed to permit fast and accurate estimation of energy and power consumption at different levels in the AADL refinement process. Power models have been built for many different hardware components (DSP, FPGA, memories, etc) as well as for software components (Ethernet layers, Inter Process Communications, etc).

Animation. ADeS (Architecture Description Simulation) [2] is a tool to simulate the behaviour of AADL models. Its purpose is to support the AADL behavioural annex, and to output graphical results of simulations. This tool makes it possible to evaluate and analyse the behaviour of a system during its specification with AADL, for instance by helping in the choice of dimensioning parameters such as scheduling, processor selection, and frequency.

Code Translation. Mappings of AADL to specific component framework and target languages have been defined, for example the AADS for the SystemC library [14].

Test Generation and Worst Case Execution Time. Taking as input a C source code, PathCrawler produces for each function a set of test cases according to generation criterion like covering all feasible paths or paths with worst case execution time [5].

3 Tool Chain on Event-B

3.1 The Event-B language

Event-B is a formal modelling method for developing systems via step-wise refinement, based on first-order logic [1, 7] Event-B models are organized in terms of two basic constructs depicted in figure 3:

– *Contexts* specify the static part of a model. They may contain carrier sets (similar to types), constants, axioms (contraining carrier sets and constants), and theorems (expressing properties derivable from axioms).
– *Machines* specify behavioural properties of the models. They may contain variables defining the state of a machine, invariants constraining that state, and events (describing possible state changes). Each event is composed of a set of guards and a set of actions. Guard state the necessary conditions under which an event may occur, and actions describe how the state variables evolve when the event occurs.

Contexts/Machines may be refined from more abstract to more concrete contexts/machines. Event-B models are systematically structured in refinement chains.

A key concept in Event-B is proof-obligation (PO) capturing the necessity to prove some internal property of the model such as typing, invariant preservation

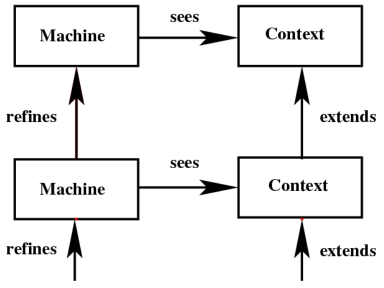

Fig. 3. Event-B Machines and Contexts

by events, and correct refinements. Strong tool support is provided in order to support this proof process.

Event-B is not specific to embedded systems design but it is currently being investigated by several industrial from different sectors (automotive, transportation, space) in the context of the DEPLOY project [6].

3.2 Tool Chain Description

The Rodin Platform is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proofs. The platform is open source and based on the Eclipse framework. It can be further extended through plug-ins and most of the Event-B related tools are provided under this form. The global tool chain is described in figure 4.

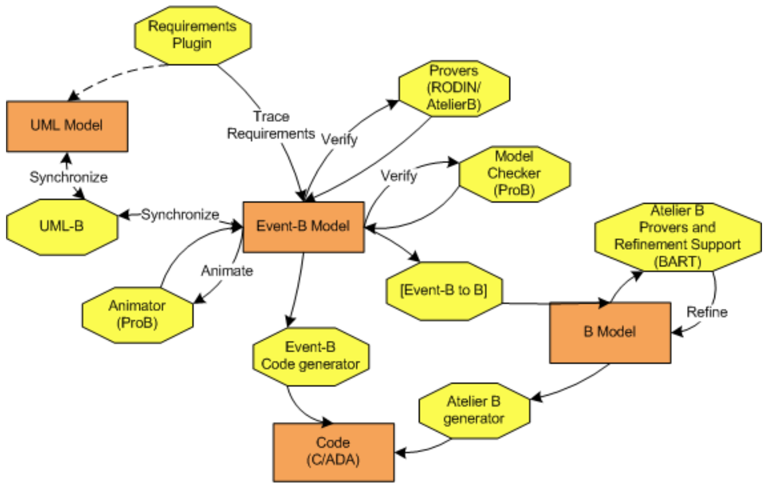

Fig. 4. Event-B Tool Chain

Requirements Engineering. Requirements developed with an external tool like DOORS or RequisitePro can be imported and linked with element models through a specific plug-in. Besides this basic traceability, a richer requirements model, based on problem frames, is also being investigated [11].

Graphical Modelling. UML-B [19] is a UML-based graphical front end for Event-B. It borrows some diagrams from UML and provides Event-B semantics to them. The class diagram is used to capture machine and contexts, while state machines are used to capture the event behaviour. An automated round-trip synchronisation is ensured with the Event-B model.

Proving. The RODIN platform has a build-in automated theorem prover for helping in the process of proving PO. In order to reach higher degree of automated proofs, it can be extended by more powerful external provers, especially those from Atelier-B, the commercial tool for B which relies on the same mathematical foundations.

Animation and Model Checking ProB [12] is an animator and model checker for the B-Method which also supports Event-B. These are two distinct functionalities are tightly related because they rely on the same underlying constrain solving engine.

The model-checker can systematically check a specification to detect consistency problems or refinement violations. It can carry out the work of the prover in a fully automated way provided the number of states is kept under control.

The animator allows fully automatic model animation by initializing the machine, checking enabled events and computing the outcome of triggered events. Animation supports the detection of invariant violation and graphical visualisation.

Code Generation Currently, code generation from Event-B models is mainly supported by going through a B tool chain using an Event-B to B translator. C and ADA code generators have been used industrially for years, especially for the design of automated metro lines [3]. Direct code generator for Event-B are also being developed.

4 Comparison and Discussion

4.1 Task Support

Unsurprisingly, the two tool chains show a number of similarities in their tool support related to similar design tasks with some differences that are highlighted

- *Requirements Activity.* Two approaches can be considered: trying to be more formal at requirements level or just trace requirements to modelled elements. The former was applied to AADL and the later to Event-B but this looks quite independent and should be more related to the gain expected from extra modelling effort at requirements level. In the case design level errors (AADL or Event-B) originated from a requirements problem, it is important to improve them.
- *Graphical Syntax.* In both cases, a graphical syntax is available to model. For AADL, it is a domain specific syntax directly reflecting concepts of embedded systems. For Event-B, it relies on the generic UML language.

- *Checks performed on the model.* Both tool chains support checks on the model. There are however more checks defined on the Event-B model and those are supported by a generic prover and model checker. In AADL, basic syntactic checks are supported by the modelling tool while more specific properties relies on a specific tool. However in Event-B such properties (such as timing, resource consumption) are more difficult to capture as they require an explicit model. In this case, the performance to proof on those properties can also be highly dependent of the way it is modelled.
- *Animator.* An animator tool is available in both tool chains. This tool is important to be able to validate the behaviour of the system with validating customers which may not be able to understand the notation (especially if completely mathematical). It is also a useful tool for the model developer to better understand its model. The AADL animator shows more powerful capabilities related to embedded systems (like impact of allocation which require a more complex explicit model in Event-B) while the Event-B animator is more powerful at computing complex animations.
- *Code generator.* Both tool chain have a link with the code level.

4.2 Domain Specific vs Generic Language

AADL is a domain specific language for the architecture of embedded systems. As such it can directly capture key elements such as timing constraints, and resource allocation. Such requirements must be explicitly modelled in Event-B and will yield a heavier, less readable and less easy to maintain models. Moreover in order to perform useful proofs, it it not trivial to design the right modelling approach, as experienced in the DEPLOY project. However once "tuned", it can be reproduced across projects.

4.3 Industrially Grown vs Carefully Designed Language

AADL was grown in an industrial context with the goal to fulfill immediate needs that were progressively raised from a communication means between people to a model that can be processed and transformed. Event-B has an opposite history with formal grounds and tools progressively opening to industrial users. This requires still on-going adaptations such as support for some missing constructs (records) and team-work (modularisation).

5 Conclusions and Perspectives

In this paper, we have presented a comparison of two recent tool chains in the context of embedded system design. Although based on quite different underlying languages, they show several similarities at task level support. As expected none is a silver bullet but they show interesting complementarities in terms of expressiveness, precision, and automation. An interesting approach would be to produce a mixed approach combining a domain specific language with a grounded but more hidden underlying formalism.

Acknowledgement

This project was financially supported by the DEPLOY FP7 project (project reference number 214158). We also warmly thanks all the partners of the SPICES ITEA consortium.

References

1. Jean-Raymond Abrial, *Modeling in Event-B, System and Software Engineering*, Cambridge University Press, 2010.
2. axlog, *ADeS*, `http://www.axlog.fr/aadl/ades_en.html`.
3. P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier, *Météor: A successful application of b in a large project*, in Proc FM '99: Wold Congress on Formal Methods (London, UK), Springer-Verlag, 1999, pp. 369–387.
4. B. Berthomieu and F. Vernadat, *Time petri nets analysis with tina*, in Proc QEST '06: 3rd Int. Conf. on the Quantitative Evaluation of Systems (Washington, DC, USA), IEEE Computer Society, 2006, pp. 123–124.
5. CEA, *PathCrawler*, `http://www-list.cea.fr/labos/fr/LSL/test/pathcrawler`.
6. DEPLOY FP7 Project, `http://www.deploy-project.eu`.
7. Event-B, `http://www.event-b.org`.
8. R.B. França, J.-P. Bodeveix, M. Filali, J.-F. Rolland, D. Chemouil, and D. Thomas, *The aadl behaviour annex - experiments and roadmap*, ICECCS, 2007, pp. 377–382.
9. Alfonso Fuggetta, *A classification of case technology*, Computer **26** (1993), no. 12, 25–38.
10. S. Gérard, P. Feiler, J-F Rolland, M. Filali, M-O Reiser, D. Delanote, Y. Berbers, Laurent Pautet, and I. Perseil, *Uml&aadl '2007 grand challenges*, SIGBED Rev. **4** (2007), no. 4, 1–1.
11. Michael Jackson, *Problem frames: analyzing and structuring software development problems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
12. M. Leuschel and M. Butler, *ProB: An Automated Analysis Toolset for the B Method*, International Journal on Software Tools for Technology Transfer **10** (2008), no. 2, 185–203.
13. Z. Manna and A. Pnueli, *The Reactive Behavior of Reactive and Concurrent System*, Springer-Verlag, 1992.
14. Univ. of Cantabria, *AADS*, `http://www.teisa.unican.es/gim/es/AADS2.php`.
15. C. Ponsard and M. Delehaye, *Towards a model-driven approach for mapping requirements on aadl architectures*, ICECCS, 2009, pp. 353–358.
16. C. Ponsard, P. Massonet, J. F. Molderez, A. Rifaut, A. van Lamsweerde, and H. Tran Van, *Early Verification and Validation of Mission Critical Systems*, Journal of Formal Methods in System Design **30** (2007), no. 3.
17. SAE, *Architecture analysis & design language (aadl)*, AS-5506, 2004.
18. Eric Senn, Johann Laurent, Emmanuel Juin, and Jean-Philippe Diguet, *Refining power consumption estimations in the component-based aadl design flow*, FDL, 2008, pp. 173–178.
19. C. Snook and M. Butler, *UML-B and Event-B: an integration of languages and tools*, The IASTED International Conference on Software Engineering - SE2008, February 2008.
20. SPICES ITEA Project, `http://www.spices-itea.org`.
21. TOPCASED, `http://www.topcased.org`.
22. A. van Lamsweerde, *Requirements Engineering. From System Goals to UML Models to Software Specifications*, Wiley, 2009.