Towards Intelligent Tool-Support for AADL Based Modeling of Embedded Systems

Dries Langsweirdt, Yves Vandewoude and Yolande Berbers

Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Leuven, Belgium
{dries.langsweirdt, yves.vandewoude, yolande.berbers}@cs.kuleuven.be

Abstract. Model-driven design (MDD) of complex embedded systems is currently based on successive cycles of model changes, analysis and simulation. This iterative process suffers from a delay between applying changes on the model and knowledge about the resulting properties of the system. Current research on Architecture Discription Languages (ADL) in general, and AADL in specific, focuses primarily on tools and support for analysis and simulation, as distinct phases during design. We give an overview of existing work on AADL, and illustrate though a case study the opportunities for a novel, integrating research domain on ADL.*

1 Introduction

Model-driven design (MDD) helps system architects master the complexities associated with the development of large and multi-concern embedded systems. Architecture Description Languages (ADL) are hereby a primary way to model the system components and their interactions. ADL specific analysis and simulation tools applied on these models can estimate the properties and the behavior of the final system, and thus predict if the system will meet its requirements. Deviations between analysis and simulation results on the one hand, and expected properties and behavior on the other, lead to model changes. As such, the design follows an iterative scheme. Changing the model implies changing the properties of the system, be it functional or non-functional. However, the relation between a model change and the resulting properties is often unclear until the subsequent analysis and simulation phase, making this iterative cycle unnecessarily long. There is a clear need for tools able to seamlessly integrate the modeling, analysis and simulation phases as to assist the architect in decision making through direct feedback on model changes. Formalization of domain knowledge and architectural patterns are prime concerns in this context. This paper identifies the need for research addressing these concerns, and does so in

* This work has been carried out as a part of the Condor project (`http://www.esi.nl` Projects->Condor) at FEI company under the responsibilities of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

the concrete context of the Architecture Analysis and Design Language (AADL) [1, 2] as a prime example of a Real-Time/Embedded (RT/E) ADL.

This paper is organized as follows. Section 2 gives an overview of the current work on AADL. Section 3 introduces an example used to pinpoint the current technical barriers, and is a first incite towards more intelligent tool-support. In section 4, we discuss our current ideas and provide a possible roadmap for future research. Conclusions are drawn in the final section.

2 Overview of current work on AADL

Four research domains applicable to AADL are currently under active development: front-end processing, code generation, analysis and simulation. Figure 1 shows a classification of the most important initiatives, which are discussed next. OSATE [3] targets front-end processing and semantic checking of AADL models, with two possibilities to extend its capabilities into the analysis and simulation domain. First, plugins can be build on top of OSATE's functionality, allowing for custom analysis on OSATE resident models. Second, models can be exported in an XMI schema, which allows for analysis on the models by external tools. The TOPCASED [4] project integrates with OSATE to visualize the AADL models. Ocarina [5] is an Ada tool suite with the ability to generate the infrastructural code in Ada or C of a distributed, real-time and high-integrity application from an AADL specification. Ocarina links the generated applications with the high-integrity middleware libraries PolyORB-HI-Ada and -C, derived from the PolyORB [6] project. STOOD [7] offers the embedded engineer multiple modeling paradigms: UML2.0, HRT HOOD and AADL 1.0. Like Ocarina, STOOD generates Ada and C code from AADL models, but the generated applications are not distributed. Cheddar [8] is a framework for schedulabilty analysis. CPN-AMI [9] is a CASE environment able to analyze and simulate Petri-net based models. Both projects are independent of AADL, but provide a good example of how the aforementioned XMI scheme (together with appropriate model transformations) can bring external analysis tools to the AADL scene. ADeS [10] aims at behavioral simulation of AADL models. The goal is to implement the entire AADL Behavior Model Annex in the simulation kernel Jimex, but currently ADeS only implements a simple behavioral model on threads. In contrast, AADS [11] transforms a subset of AADL to SystemC for simulation. Building on the SCoPE [12] project, AADS is well suited for HW/SW co-design. Finally, [13] proposed execution of AADL models based on a translation to the synchronous languages Scade and Lustre.

Only minor integration between the different domains is noted. Intelligent integration of the available work to assist the system architect during the actual act of modeling is absent, and is why we suggest a fifth domain on design support (see figure 1). Intelligent in this context refers to the availability of formalized knowledge and patterns specific to the embedded domain. Reasoning algorithms could apply this knowledge on the concrete, but potentially incomplete, models

to deduct architectural suggestions, warnings and optimal properties. A more concrete discussion can be found under section 4.

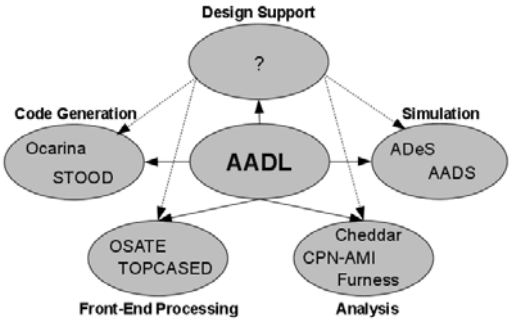


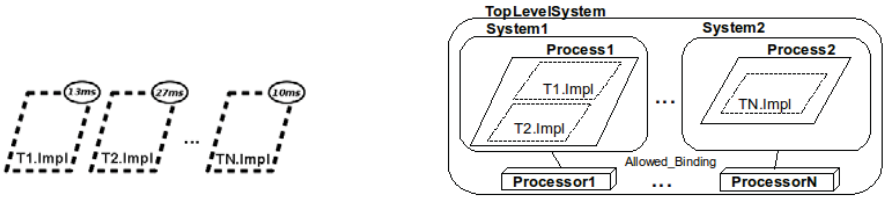
Fig. 1: Overview of the four existing and proposed fifth domain on AADL.

3 Case Study

We present the extension of an existing OSATE plugin as an illustration of more integrated modeling support. It also identifies current problems with analysis on unfinished, declarative AADL models.

3.1 System Specification Versus Instantiation

AADL differentiates between system specification and instantiation. A system is defined as completely instantiable if: "the system implementation being instantiated is completely specified and completely resolved". The tools presented in section 2 almost exclusively work on instantiated models, and thus complete with respect to the compositional and legality rules of AADL. They are incapable of gathering information from, and act on, declarative models.



(a) (b)

Fig. 2: Automated model completion through bin-packing and scheduling from (a) the isolated thread set, to (b) the instantiable system implementation.

3.2 Automated Bin-packing and Scheduling

The case study extends OSATE's bin-packing and scheduling plugin, which is an implementation of the work done on partitioned bin-packing algorithms by de Niz et al. [14, 15]. The plugin automates the assignment of threads to processors available in an instantiated AADL model. We extended the plugin to work on declarative models as well. Provided with a minimum of information (a thread set and its properties, with optionally a processor and/or bus used as templates), our plugin calculates the amount of needed processors and links to make the threads schedulable. The plugin then extends the declarative model bottom-up, based on the outcome of the analysis, to a completely instantiable model with bounded threads. This model transformation is illustrated in figure 2.

3.3 OSATE Deficiencies and AADL Intricacies

Although the case study is limited, we noted the following four interesting problems. First, the API of OSATE with respect to extensive manipulations of declarative models could be much improved upon. One example is the asymmetry between addition and removal of component types or implementations. For removal, the programmer needs to rely directly on the Eclipse Ecore infrastructure. Second, because of the AADL semantics, information can be scattered throughout the declarative model in complex ways. Instantiated models do not have this problem as such, because the relations between components are fixed and their properties can be referenced directly. Gathering information on the declarative model quickly results in multiple model scans, potentially leading to a scalability issue in the current OSATE implementation. Finally, two forms of ambiguity in the use of the AADL standard (first noted by Delanote in [16]) complicate the automated extension and analysis of an unfinished model. First, the legality rules are different for each component category, making component composition not only complex, but also ambiguous. For example, the model extension in the case study wraps each process in a separate system component. There are however multiple other legal ways with respect to the standard to complete the model, without the advantages of one approach over another being obviously clear. Second, there is no well defined relation between the analysis of certain system properties, and the AADL model properties needed to conduct it. With these relations being unclear, it becomes hard to discover missing information in the model.

4 Roadmap

As mentioned in section 2, key to assisted modeling is formalization and embedding of domain specific knowledge and architectural patterns, together with appropriate reasoning algorithms, in the tools the system architect uses to construct ADL models. Concrete properties and interconnections of model components can as such be linked with corresponding analysis and simulation routines.

The results of these routines, automatically invoked on each incremental model change, can be fed back to the architect in the form of suggestions on architectural changes. If appropriate, changes can be performed automatically by the tool, already illustrated in the case study. Note that, as stated in section 3.1 and 3.3, analyzing an incomplete model is a non-trivial task making analysis and simulation adaptation, or virtual model completion, a necessity. The first challenge, formalization, is in principle ADL-agnostic, and depends on the RT/E domain in general. Harvesting model properties and performing model changes on the other hand, depend on the legality and compositional rules of the concrete ADL. Both challenges are interesting tracks that can be addressed by the proposed domain on design support.

5 Conclusion

This paper discusses the need for more research focusing on support for the embedded systems architect during modeling, compared to analysis of a certain made choice. Through an AADL case study, we identified some of the existing barriers and defined a roadmap for future research.

References

1. Feiler, P., Gluch, D., Hudak, J.: AADL: An Introduction. Tech. rep., SAE (2006)
2. SAE: Architecture Analysis & Design Language (AS5506A), `http://www.sae.org`
3. Open Source AADL Tool Environment (OSATE). Techn. rep., SEI (2006)
4. The Open-Source Toolkit for Critical Systems, `http://www.topcased.org/`
5. Hugues, J., Zalila, B., Pautet, L.: From the Prototype to the Final Embedded System Using the Ocarina AADL Tool Suite. In: ACM TECS 7 No.4, Art.42 (2008)
6. Vergnaud, T., Hugues, J., Pautet, L.: PolyORB: A schizophrenic middleware to build versatile reliable distributed applications. LNCS, vol. 3063, pp 106–119. Springer, Heidelberg (2004)
7. Ellidiss-Software: STOOD, `http://www.ellidiss.com/stood.shtml`
8. Singhoff, F., Legrand., J., Tchamnda, L.: Cheddar: A flexible real time scheduling framework. J. ACM Ada Lett. 24, 1–8 (2004)
9. The CPN-AMI home page, `http://www.lip6.fr/cpn-ami`
10. ADeS: a simulator for AADL, `http://www.axlog.fr/aadl/ades_en.html`
11. Varona-Gmez, R., Villar, E.: AADL Simulation and Performance Analysis in SystemC. In: IEEE ICECCS, pp. 323–328. IEEE Computer Society, Potsdam (2009)
12. SCoPE v1.0.0 UC 2008, `http://www.teisa.unican.es/scope`
13. Jahier, E., Halbwachs, N., Lesens, D.: Virtual execution of AADL models via a translation into synchronous programs. In: Proceedings of the 7th ACM&IEEE ICESS, pp 134–143. ACM, NY (2007)
14. de Niz, D., Rajkumar, R.: Partitioning Bin-Packing Algorithms for Distributed Real-Time Systems. I. J. of Embedded Systems 2 No.3/4, 196–208 (2006)
15. de Niz, D., Bhatia, G., Rajkumar, R.: Model-Based Development of Embedded Systems: The SysWeaver Approach. In: 12th IEEE RTAS, pp. 231–242. IEEE Computer Society, San Jose (2006)
16. Delanote, D., Van Baelen, S., Joosen, W., Berbers, Y.: Using AADL in Model Driven Development. UML&AADL'2007, ICECCS07. IEEE, Auckland (2007)