A Model Checker for AADL[⋆]

Marco Bozzano[2], Alessandro Cimatti[2], Joost-Pieter Katoen[1],
Viet Yen Nguyen[1], Thomas Noll[1], Marco Roveri[2], and Ralf Wimmer[3]

[1] RWTH Aachen University, Germany
[2] Fondazione Bruno Kessler, Trento, Italy
[3] Albert-Ludwigs-University Freiburg, Germany

Abstract. We present a graphical toolset for verifying AADL models, which are gaining widespread acceptance in aerospace, automobile and avionics industries for comprehensively specifying safety-critical systems by capturing functional, probabilistic and hybrid aspects. Analyses are implemented on top of mature model checking tools and range from requirements validation to functional verification, safety assessment via automatic derivation of FMEA tables and dynamic fault trees, to performability evaluation, and diagnosability analysis. The toolset is currently being applied to several case studies by a major industrial developer of aerospace systems.

1 Introduction

System-level languages like Architecture Analysis and Design Language (AADL) and SysML are increasingly adopted by industry for designing new safety-critical systems. Their added advantage is that they enable the system designer to capture the elusive interaction between hardware and software. In particular nominal and degraded modes of operation, the propagation of faults between subsystems, and the mechanisms for the system to recover from them are essential to a comprehensive system-level design.

As part of the COMPASS project [1] (Correctness, Modelling and Performance of Aerospace Systems) we developed a formal semantics for AADL (briefly discussed in Sect. 2) that incorporates functional, probabilistic and hybrid aspects [5]. This is fundamental for tool-supported formal verification. Over the past two years we have built a graphical toolset, called the COMPASS platform (see Sect. 3), supporting AADL and based on model checking techniques to verify them. The tool is currently being applied to several case studies by a major industrial developer of aerospace systems [4].

2 Specification Language

To make AADL amenable to formal verification, we have cut out its superfluous features and added support for hybrid aspects. The tool's resulting input language thus follows the component-based paradigm. It supports both software (e.g., processes and threads) and hardware components (e.g., memories and processors) as first-class objects. Each component is given by its type, describing the interface, and its implementation, describing the interactions via a finite state automaton. Sets of interacting

[⋆] Funded by ESA/ESTEC under Contract No. 21171/07/NL/JD

components can be grouped into composite components, enabling the modeler to manage the system's complexity by introducing a component *hierarchy*. Communication is achieved via exchange of messages on event ports, in a rendez-vous manner. Moreover, components may exchange data through typed data ports (e.g., bool, integer and real data types). Timed and hybrid behavior can be expressed by means of real-valued variables with (linear) time-dependent dynamics.

The resulting hierarchical system model, also referred to as *nominal model*, describes the system behavior under normal operation. This is complemented by an *error model* which expresses how the system can fail. Moreover, a subset of the nominal components may be designated as dealing with error diagnosis and recovery; they are referred to as FDIR (Fault Detection, Identification and Recovery). The error model expresses how faults may affect normal operation and may lead the system into a degraded mode of operation. It is modeled as a (probabilistic) finite state automaton, where transitions may occur due to error events which may be annotated with a rate that indicates the expected number of occurrences per time unit. Transitions can also occur because of error propagations from other components. The nominal and error models are linked through a so-called *fault injection*. A fault injection expresses the effect of the occurrence of the corresponding error on the nominal model. Multiple fault injections are possible. The process of integrating the nominal models with the error models and the fault injections, is called *model extension* [8]. Finally, in order to enable modeling of partial observability and analysis of FDIR components, our language allows the modeler to explicitly define a set of observables.

We refer to [5–7] for a more detailed description of the tool's input language, a discussion of the similarities and extensions with respect to AADL, and a simple example (a processor failover system). In particular, [5] presents a formal semantics for all the language constructs, based on networks of event-data automata (NEDA).

3 Toolset

The COMPASS platform [7] is an integrated toolchain, based on state-of-the-art tools and symbolic model checking techniques, for verification and validation of AADL models. It builds upon the NuSMV [16] symbolic model checker, the MRMC [15] probabilistic model checker, the Sigref [18] bisimulation minimization tool, and the RAT [17] requirements analysis tool. The architecture of the tool is sketched in Figure 1. It refers to two inputs, namely the SLIM model (our extended variant of AADL) and property patterns. The latter describe properties, expressed in the user-friendly patterns by Grunske [14] and Dwyer [12], which are converted to respectively CSL [3], LTL and CTL formulae. These inputs are processed into the lower-level formalisms of NuSMV and MRMC. A set of visualizers transforms the output (like counterexample traces and fault trees) back to the user. Feature-wise, the toolset supports the following analyses.

Requirements Validation, implemented by RAT, focuses on assessing the quality of a set of requirements with respect to the user expectations, before a model of the actual system is built. It is possible to check that a set of properties is logically consistent, and that it is strict enough but not too strict, by checking for compatibility with a set of possibilities, and for logical consequence of a set of assertions.

Functional Verification comprises random and user-guided simulation, deadlock detection, and verification of functional properties via model checking. The result can be a statement that a property holds, or a counterexample or witness trace, in case the property is refuted or for simulation. This analysis is based on NuSMV, which supports BDD-based, SAT-based, and (for hybrid systems) SMT-based model checking.

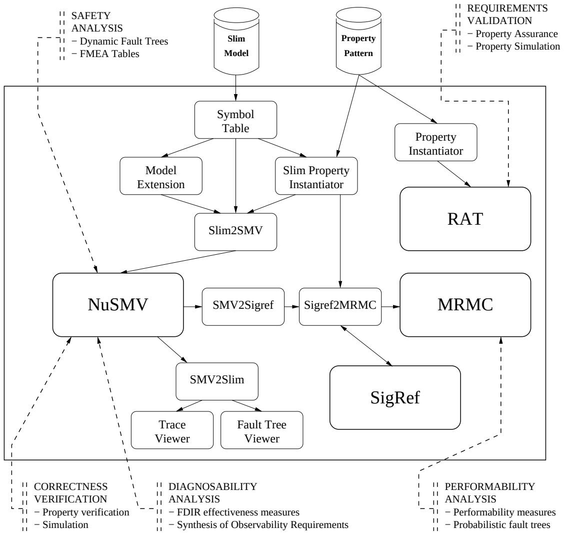

Fig. 1. Architecture of the toolset.

Safety Analysis comprises traditional techniques for hazard analysis, such as (Dynamic) Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA), that are used to assess system behavior in presence of faults. FTA constructs all possible chains of basic faults, represented as a tree, that may be responsible for an undesired behavior. FMEA is similar, but starts from a set of faults and analyzes the impact on a set of properties.

Diagnosability Analysis checks whether a system is diagnosable with respect to a user-specified property, that is, whether an ideal diagnoser has enough observations to identify the set of causes of a specific faulty behavior.

Fault Detection, Isolation and Recovery (FDIR) focus on, respectively, verifying whether a given fault can be properly detected, isolated and recovered.

Performance Evaluation computes, using a probabilistic property, system performance under degraded operations. It furthermore includes the evaluation of fault trees by computing the probability of the top event. These analyses are based on MRMC.

The toolset is being extensively evaluated on a set of industrial-size case-studies [4]. Moreover, it is being tuned to achieve optimal performance. Preliminary experiments indicate that the choice of the verification technique (e.g., BDD-based versus SAT-based) may be important to achieve a good performance and scalability for the different types of analyses. The details of those experiments can be found on the project's website [1].

4 Related Work and Conclusions

In this paper we have presented a comprehensive toolset for the verification and validation of AADL models. The toolset supports several analyses ranging from requirements validation to functional verification, safety analysis, diagnosability and performance evaluation. It is available under an open source license via the COMPASS website [1].

Several tools have been developed to analyse AADL specifications, however none of them provides support for all analysis described in this paper. We mention AADL2BIP [11], which translates AADL models into a formalism called BIP, and is able to perform simulation and deadlock detection. ADeS [2] is a software tool to simulate the behaviour of system architectures described in AADL. This tool is mainly a simulator that allows for the evaluation and analysis of system behavior, with no formal analysis underneath. Finally, Cheddar [10] and the Furness^{TM} Toolset [13] support only schedulability analysis.

Due to constraints on paper-length, this paper only describes an overview of the COMPASS toolset. An in-depth discussion of the toolset, the formal semantics of AADL and the relation to other works can be found in our journal paper [9].

References

1. The COMPASS project. `compass.informatik.rwth-aachen.de/`.
2. ADeS, a simulator for AADL. `http://www.axlog.fr/aadl/ades_en.html`.
3. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. on Soft. Eng.*, 29(6):524–541, 2003.
4. M. Bozzano, R. Cavada, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and X. Olive. Formal Verification and Validation of AADL Models. In *Proc. ERTS 2010*. To be published.
5. M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. Codesign of Dependable Systems: A Component-Based Modelling Language. In *Proc. MEMOCODE'09*, pages 121–130. IEEE, 2009.
6. M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. Model-based codesign of critical embedded systems. In *Proc. ACES-MB'09*, pages 87–91, 2009.
7. M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In *Proc. SAFECOMP 2009*, pages 173–186. Springer, 2009.
8. M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *Int. J. on Software Tools for Technology Transfer*, 9(1):5–24, 2007.
9. Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal*, doi: 10.1093/com, March 2010.
10. Cheddar: a free real time scheduling tool. `https://wiki.sei.cmu.edu/aadl/index.php/Cheddar`.
11. M.Y. Chkouri, A. Robert, Bozga M, and J. Sifakis. Translating AADL into BIP – application to the verification of real-time systems. In *Proc. ACES-MB'08*, pages 39–53. Springer, 2008.
12. M. Dwyer, G.S. Avrunin, and J.C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. ICSE'99*, pages 411–420. IEEE, 1999.
13. The Furness$^{\text{TM}}$ Toolset. `http://www.furnesstoolset.com/`.
14. Lars Grunske. Specification patterns for probabilistic quality properties. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *ICSE*, pages 31–40. ACM, 2008.
15. MRMC – Markov Reward Model Checker. `http://www.mrmc-tool.org/`.
16. The NuSMV Model Checker. `http://nusmv.fbk.eu`.
17. RAT – Requirements Analysis Tool. `http://rat.fbk.eu`.
18. R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker. Sigref – A Symbolic Bisimulation Tool Box. In *Proc. ATVA'06*, pages 477–492. Springer, 2006.