Model-Based Codesign of Critical Embedded Systems[⋆]

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

[1] Fondazione Bruno Kessler, Italy
{bozzano,cimatti,roveri}@fbk.eu
[2] RWTH Aachen University, Germany
{katoen,nguyen,noll}@cs.rwth-aachen.de

Abstract. We present a comprehensive methodology for the specification and analysis of critical embedded systems. The methodology is based on an architectural design language that enables modeling of both software and hardware components, timed and hybrid behavior, faulty behavior and degraded modes of operation, error propagation and recovery. The methodology is supported by an integrated platform, implemented on top of state-of-the-art tools, that provides verification capabilities ranging from requirements analysis to functional verification, safety assessment, performability evaluation, diagnosis and diagnosability.

1 Introduction

The design of critical embedded systems is a very complex and highly challenging task, for a number of reasons. First, it requires designing and assembling heterogeneous components, implemented either in hardware or in software, and their interactions. Secondly, it has to take into account functional requirements as well as several sorts of non-functional requirements, such as (real-)time constraints, performability and safety requirements.

In this paper we present a comprehensive and tool-supported methodology for the design of critical systems, following the component-based paradigm. Component-based design helps to master design complexity while, at the same time, allowing for reusability. The key principle is a clear distinction between component behavior (implementation) and the interactions between the individual components (interfacing). The internal structure of a component implementation is specified by its decomposition into subcomponents, together with their hardware/software bindings and their interaction via connections over ports.

The design methodology is built on top of the SLIM modeling language, an architectural language inspired by SAE's AADL [9] (Architecture Analysis and Design Language) and the related Error Model Annex [10]. SLIM inherits the most important features of AADL, such as multiway communication, dynamic

[⋆] Funded by ESA/ESTEC under Contract No. 21171/07/NL/JD reconfiguration of components and port connections, and probabilistic error behavior and propagation, while enriching it with constructs to express timed and hybrid behavior. Moreover, the SLIM language is endowed with a formal semantics that cover all of its aspects in a clear and unambiguous way [3].

The methodology proposed in this work is targeted at the architectural design of critical embedded systems, and in particular it covers modeling and verification of the following aspects: requirements analysis, verification of functional correctness, safety assessment and fault tolerance measures, quantitative and performability analysis, and partial observability analysis, including effectiveness of the FDIR (Fault Detection, Identification and Recovery) components.

The proposed approach is being investigated in the COMPASS project[3] (Correctness, Modeling, and Performance of Aerospace Systems) in the aerospace domain, and results as a response to an invitation to tender by the European Space Agency. The techniques described in this work, however, are applicable in general to every domain where design of critical embedded systems is involved.

The paper is structured as follows. In Section 2 we describe the main features of the SLIM language; in Section 3 we give an overview of the methodology; in Section 4 we discuss the COMPASS tool, implementing the methodology, and finally we draw some conclusions and discuss future directions in Section 5.

2 The SLIM Language

The SLIM language follows the component-based paradigm. In SLIM, it is possible to refer to both software (e.g. threads and processes) and hardware components (e.g. memories and processors) as first-class objects. Each component is given via its type, describing the interface, and its implementation, describing the interactions via a finite state automaton. Sets of interacting 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

[3] http://compass.informatik.rwth-aachen.de

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* [4]. Finally, in order to enable modeling of partial observability and analysis of FDIR components, the SLIM language allows the modeler to explicitly define a set of observables.

We refer to [3] for a more detailed description of the language, a discussion of the similarities and extensions with respect to AADL, and a simple example (a processor failover system). Moreover, [3] presents a formal semantics for all the language constructs, based on networks of event-data automata (NEDA).

3 Methodology

The methodology discussed in this paper is inspired by the framework described in [1], which provides a unifying view of different aspects of system engineering, within the context of model checking. In order of increasing complexity, the first problem that we consider is system functional correctness. Functional requirements are traditionally expressed in temporal logic, e.g. Computation Tree Logic (CTL) or Linear Temporal Logic (LTL). Technologically, model checking techniques are used to exhaustively explore every possible system behavior, providing a formal guarantee that a given requirement is obeyed.

Safety analysis investigates the behavior of a system in degraded conditions, that is, when some parts of the system are not working properly due to malfunctions. It includes hazard analysis, whose goal is to identify all the hazards of the system and ensure that the system meets the safety requirements that are required for its deployment and use. Examples of hazard analysis techniques are Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA). Model-based safety analysis is in turn based on model checking techniques [4].

Quantitative analysis and performability aim at evaluating system performance with respect to timed and probabilistic requirements. They also include probabilistic versions of safety and diagnosability measures. The related requirements can be expressed in Continuous Stochastic Logic (CSL). The implementation of these analyses is based on probabilistic model checking techniques.

Diagnosis can be seen as the problem of safety analysis carried out at run-time. It is usually performed on systems which provide limited run-time sensing, and under the hypothesis of partial observability. Diagnosis starts from the observed run time behavior of a system, and tries to provide an explanation (in terms of hidden states). In particular, diagnosis is often the problem of identifying the set of possible causes of a specific unexpected or faulty behavior. Probabilistic information can be taken into account, in order to search for the most likely explanation. Another related problem is diagnosability, i.e., the analysis, at design time, of diagnosis capabilities. Finally, the problem of synthesis consists in the automatic generation of controllers from specifications. The latter problem has been tackled by planning techniques based on model checking.

Finally, requirements validation is used to check correctness and completeness of a set of properties. Requirements validation is performed before the system architectural design starts, and has the goal of ensuring the quality of system requirements. In particular, our approach enables checking for logical consistency, i.e., freedom from contradictions. Moreover, it is possible to check whether a given set of properties is strict enough to rule out unwanted behavior, and not too strict to disallow for certain desirable behavior.

4 Tool Support

The methodology is supported by an integrated toolset, which is built on top of existing state-of-the-art tools for formal verification, based on model checking. In particular, the toolset builds upon the NuSMV [7] symbolic model checker, the MRMC [6] probabilistic model checker, and the RAT [8] requirements analysis tool. The architecture of the tool set is shown in Fig. 1. The toolset takes as input a model written in the SLIM language, and a set of property patterns, used to instantiate formal requirements. Depending on the context, instantiated properties are expressed in CTL, LTL or CSL temporal logics.
A few building blocks take care of performing *model extension*, translating the SLIM input model into NuSMV and MRMC formats when needed, and visualize traces and fault trees.

The following analyses are supported. Requirements validation is used to analyse the quality (correctness and completeness) of the requirements, and is carried out by the RAT tool. Correctness verification focuses on verification of functional requirements, and is implemented on top of NuSMV; NuSMV implements standard symbolic model checking techniques such as BDD-based and SAT-based (bounded) model checking, as well as SMT (Satisfiability Modulo Theory)-based techniques to deal with hybrid models. Safety analysis supports two of the most popular hazard analysis techniques, namely FTA and FMEA, that are carried out by FSAP [5], a plugin of NuSMV. Diagnosability analysis focuses on the evaluation of the effectiveness of the FDIR

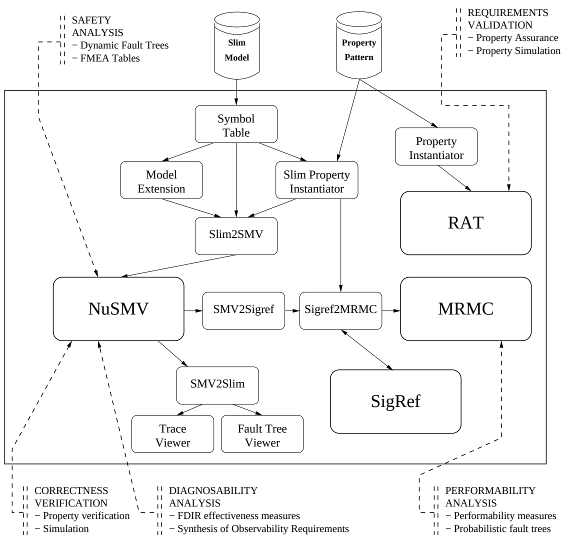

Fig. 1. Architecture of the toolset.

sub-system; these functionalities are built on top of NuSMV and FSAP. Finally, performability analysis evaluates a SLIM model with respect to probabilistic requirements; it is implemented on top of MRMC. For more information on the toolset, its architecture, and the analyses that are supported, we refer to [2].

5 Conclusions

In this paper we have presented a comprehensive methodology and a toolset for the specification and analysis of critical embedded systems, that focuses on system features such as (real-)time and faulty behavior, degraded modes of operation, diagnosis and performability. The methodology and toolset are currently being evaluated on industrial-size case studies from the aerospace domain, that will provide a substantial insight into their applicability and effectiveness.

Our methodology is applicable to any domain where, e.g., timing, system performance and safety are at stake. Examples are avionics, transportation, including railways and automotive, power plants, and the medical domain. Our approach is based on a general purpose architectural language, and it is especially targeted at modeling and analyzing systems designs at the architectural level. It can be complemented by specific implementation-level languages to deal with the most implementation-oriented features of system design.

The toolset is under active development and evaluation. A thorough experimental evaluation is planned, based on a comprehensive set of case studies.

Finally, some of the modifications to the AADL language that have been incorporated into SLIM, have been brought into the AADL standardization bodies for evaluation and proposed as a possible extension of the standard.

References

1. P. Bertoli, M. Bozzano, and A. Cimatti. A Symbolic Model Checking Framework for Safety Analysis, Diagnosis, and Synthesis. In *Model Checking and Artificial Intelligence*, volume 4428 of *LNCS*, pages 1–18. Springer, 2007.
2. 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'09*. Springer, 2009.
3. M. Bozzano, A. Cimatti, V. Y. Nguyen, T. Noll, J. P. Katoen, and M. Roveri. Codesign of Dependable Systems: A Component-Based Modeling Language. In *Proc. MEMOCODE '09*, 2009.
4. M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, 2007.
5. The FSAP/NuSMV-SA platform. `http://sra.fbk.eu/tools/FSAP`.
6. The MRMC model checker. http://wwwhome.cs.utwente.nl/ zapreevis/mrmc/.
7. The NuSMV model checker. `http://nusmv.fbk.eu`.
8. RAT: Requirements Analysis Tool. `http://rat.fbk.eu`.
9. Architecture Analysis and Design Language (AADL) V2. SAE Draft Standard AS5506 V2, International Society of Automotive Engineers, Mar. 2008.
10. Architecture Analysis and Design Language Annex (AADL), Volume 1, Annex E: Error Model Annex. SAE Standard AS5506/1, SAE International, June 2006.